



# **Spanning edge betweenness for large graphs and percolation**

**Guilherme Cristóvão e Silva Ribeiro**

Thesis to obtain the Master of Science Degree in

**Computer Science and Engineering**

Supervisors: Prof. Pedro Tiago Gonçalves Monteiro  
Prof. Andreia Sofia Monteiro Teixeira

**Examination Committee**

Chairperson: Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur  
Supervisor: Prof. Pedro Tiago Gonçalves Monteiro  
Member of the Committee: Prof. Pedro Ribeiro

**October 2022**



# Acknowledgments

I would like to thank my parents for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible. Thank you for educating me to always believe in myself, that everything is possible to achieve, through hard work and patience. You helped me unconditionally whenever I struggled, and thanks to your effort and dedication I always came out stronger. Everything I achieved in life, I achieved because of both of you, who guided me and supported me in every step. The man I am today could not exist without you.

To my sister, I would like to thank you for all the headaches, they kept the blood pumping through my head. I also wanna thank my grandparents, aunts, uncles and cousin for their understanding and support throughout all these years.

I would also like to thank my girlfriend, Catarina. Thank you for your unconditional support throughout all these hard years. It has been a difficult ride but you made it easier for me. Thank you for your patience and love. Thank you for always making me laugh, and for cheering me up, whenever necessary.

I would also like to acknowledge my dissertation supervisors Prof. Pedro Monteiro and Prof. Sofia Teixeira for their insight, support and sharing of knowledge that has made this Thesis possible. A word of appreciation to Prof. Alexandre Francisco, whose collaboration was also fundamental for this work.

Last but not least, to all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.

To each and every one of you – Thank you.



# Abstract

Centrality measures are a vital tool to study networks. After a new centrality measure is introduced, it is important to be part of studies, to become relevant in the community. Spanning Edge Betweenness was recently introduced, and this is one of those studies. An overview of network science concepts and centrality measures is provided. A brief explanation of Spanning Edge Betweenness' theoretical concepts can also be seen in this work. The implementation options can also be consulted along with practical experimentations to try to minimize the computational time cost of calculating such centrality measure. These experimentations consist of either taking advantage of the parallelizable computation algorithm, or preprocessing algorithms that could positively impact the calculation time. We achieved promising results for both alternatives. Another method to reduce the computation time could be the calculation of the approximated centrality values. This has been previously done for Spanning Edge Betweenness in unweighted networks. In this work, we propose an implementation for this approximation for weighted networks. This method also achieves promising results regarding the improvements in computation time. Finally, we perform edge percolation, comparing Spanning Edge Betweenness to Edge Betweenness regarding five metrics. For each centrality measure, we introduce two variations, one where there is no recalculation between every edge removal, and one where there is. We observed significant differences between both variations and between both centrality measures. This work makes Spanning Edge Betweenness available to be used for large-scale graphs, while also contributing to a bigger understanding of said measure, regarding the connectivity of a network.

## Keywords

Spanning Edge Betweenness; Approximate centrality measures; Edge percolation.



# Resumo

As medidas de centralidade são uma ferramenta vital para estudar redes. Após a introdução de uma nova medida de centralidade, é importante que esta faça parte de estudos, para se tornar relevante na comunidade. Spanning Edge Betweenness foi introduzida recentemente, e este é um desses estudos. Uma visão geral dos conceitos de ciência das redes e medidas de centralidade é fornecida. Uma breve explicação dos conceitos teóricos da Spanning Edge Betweenness também pode ser vista neste trabalho. As opções de implementação podem ser consultadas juntamente com experiências práticas para tentar minimizar o custo computacional do tempo de cálculo desta mesma medida. Essas experiências consistem em aproveitar o algoritmo de computação paralelizável ou algoritmos de pré-processamento que podem ter um impacto positivo no tempo de cálculo. Alcançamos resultados promissores para ambas as alternativas. Outro método para reduzir o tempo de computação poderia ser o cálculo dos valores de centralidade aproximados. Isso foi feito anteriormente para Spanning Edge Betweenness em redes não pesadas. Neste trabalho, propomos uma implementação desta aproximação para redes pesadas. Este método também alcança resultados promissores em relação às melhorias no tempo de computação. Por fim, realizamos a percolação de arestas, comparando Spanning Edge Betweenness com Edge Betweenness em relação a cinco métricas. Para cada medida de centralidade, introduzimos duas variações, uma onde não há recálculo entre cada remoção e outra onde há. Observamos diferenças significativas entre as duas variações e entre as duas medidas de centralidade. Este trabalho disponibiliza a Spanning Edge Betweenness para uso em grafos de grande escala, além de contribuir para um maior entendimento desta medida, no que diz respeito à conectividade de uma rede.

## Palavras Chave

Spanning Edge Betweenness; Medidas de centralidade aproximadas; Remoção de arestas.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Objectives . . . . .	3
1.2	Outline . . . . .	4
<b>2</b>	<b>Network Science Background</b>	<b>5</b>
2.1	Network Science Concepts . . . . .	7
2.1.1	Graph Theory . . . . .	7
2.1.2	Graph Representations . . . . .	8
2.1.3	Network Models . . . . .	9
2.2	Centrality Measures and Connectivity . . . . .	12
2.2.1	Centrality Measures . . . . .	12
2.2.2	Libraries . . . . .	17
2.2.3	Robustness . . . . .	17
2.3	Related work . . . . .	18
2.3.1	Networks . . . . .	18
2.3.2	Nodes vs Edges . . . . .	19
2.3.3	Approximating Centrality Measures . . . . .	20
2.3.4	Robustness . . . . .	20
<b>3</b>	<b>Spanning Edge Betweenness – Implementation and Parallelism</b>	<b>25</b>
3.1	Theory . . . . .	27
3.1.1	Determinants . . . . .	27
3.2	Implementation . . . . .	28
3.3	Results . . . . .	31
<b>4</b>	<b>Spanning Edge Betweenness – Weighted Approximation</b>	<b>39</b>
4.1	Algorithm . . . . .	41
4.2	Results . . . . .	43

<b>5</b>	<b>Edge Percolation</b>	<b>47</b>
5.1	Number of Connected Components . . . . .	50
5.2	Size of the Giant Connected Component . . . . .	52
5.3	Diameter of the Giant Connected Component . . . . .	54
5.4	Median of Connected components' size . . . . .	57
5.5	Average path length inside the giant connected component . . . . .	59
5.6	Number of connected components and the giant's connected component size . . . . .	61
5.7	Conclusions for each network . . . . .	63
<b>6</b>	<b>Conclusion</b>	<b>65</b>
6.1	Future Work . . . . .	67
	<b>Bibliography</b>	<b>69</b>

# List of Figures

2.1	Circles represent nodes and lines represent edges. Edges in Figure 2.1a are bidirectional, meaning they connect both nodes to one another in both ways, while in Figure 2.1b every edge is unidirectional connecting one node to another, just in that specific way, pointed by the arrow. In red we have the shortest path between nodes <i>A</i> and <i>D</i> . . . . .	7
2.2	Figure 2.2a represents a directed weighted graph, where a numerical value is attributed to every edge. This values can have multiple different meanings, for instance it could represent the flow of a water system. In Figure 2.2b we can see the shortest path between nodes <i>A</i> and <i>D</i> , different from Figure 2.1b because of the addition of edges' weights. . .	8
2.3	Figures 2.3a, 2.3b, 2.3c and 2.3d are visual examples of the representations presented above for the graph from Figure 2.1b. . . . .	10
2.4	Image retrieved from [1], illustrating correlation between different studied centrality measures. A color scale was used, where green represents less similarity and red represents more similarity. The three similarity clusters are: betw-pr15, katz-eig and deg-cc. . . . .	19
2.5	Image retrieved from [2], illustrating edge percolation on twelve different networks with seven different structural measures. Each plot has two parts, the top one for the fraction of nodes belonging to the largest component and the bottom one for the normalized susceptibility. . . . .	24
3.1	Real world networks calculation times with multiple different <i>PySpark</i> configuration. <i>w</i> : worker nodes; <i>c</i> : cores. . . . .	37
5.1	Evolution of the number of connected components with the removal of edges, for every real world network studied. . . . .	51
5.2	Evolution of the size of the giant connected component with the removal of edges, for every real world network studied. . . . .	53
5.3	Evolution of the giant's connected component diameter with the removal of edges, for every real world network studied. . . . .	56

5.4	Evolution of the median of the connected components' size with the removal of edges, for the real world networks studied. . . . .	58
5.5	Evolution of the average path length inside the giant connected component with the removal of edges, for the real world networks studied. . . . .	60
5.6	Relation between the evolution of the number of connected components with the evolution of the giant's connected component size, for all the real world networks studied. . . . .	62

# List of Tables

2.1	In this table, there is more information regarding each centrality measure previously presented. <b>Edges/Nodes</b> , <b>Directed/Undirected</b> and <b>Weighted/Unweighted</b> columns represent if a measure is exclusive to any of the categories or not. <b>Complexity</b> represents the asymptotic worst case of calculating the measure to every structure in the network. <b>Definition</b> is a reference to where the measure was defined/presented. <b>Comparisons</b> references the papers where measures present in this table were compared, mainly regarding percolation. In those papers, there may be measures that are not in this table/work. <b>Libraries</b> represents if the measure is present on NetworkX (NX), iGraph, or in both or neither one of these Python libraries . . . . .	16
3.1	Real world networks' characteristics . . . . .	32
3.2	Time that different decompositions take to calculate SEB for each network, in seconds. "scipy's decomposition" stands for the native linear algebra function, and "lapack's decomposition" represents the function from the LAPACK module inside Scipy. . . . .	32
3.3	Real world networks SEB calculation time without and with preprocessing applied, time in seconds. . . . .	33
3.4	SEB calculation time for networks generated from the Newman-Watts-Strogatz model with and without preprocessing, time in seconds. n:number of nodes; k:joined with k nearest neighbours; p:probability of adding a new edge. Networks generated using the NetworkX package. . . . .	34
3.5	SEB calculation time for networks generated from the Barabási–Albert model with and without preprocessing, time in seconds. n:number of nodes; m:number of nodes to attach from a new node. Networks generated using the NetworkX package. . . . .	35
3.6	Comparison between original serial calculation times to the use of PySpark, for each network. Time in seconds . . . . .	35

4.1 Comparison of times between approximated and exact SEB values, time in seconds.  $\epsilon =$   
0.01 and  $\beta = 1$ . . . . . 43

# List of Algorithms

- 3.1 Weighted Network . . . . . 28
- 3.2 Unweighted Network . . . . . 29
- 3.3 Calculate Determinant . . . . . 29
- 3.4 Unweighted Preprocessing . . . . . 30
- 3.5 Weighted Preprocessing . . . . . 30
- 3.6 PySpark SEB calculation . . . . . 31
- 4.1 SEB approximations for unweighted networks. . . . . 41



# 1

## Introduction

### Contents

---

1.1 Objectives . . . . .	3
1.2 Outline . . . . .	4

---



Networks can represent everything in our life. They can describe the water infrastructure of a given city, the air traffic between airports in different countries, or even the daily interactions between work colleagues. This constant presence in our lives led to an interest in studying these networks and their properties for multiple different purposes.

Graphs are one of the simplest ways of representing these networks. Nodes represent entities, and edges represent the interactions/connections between them. There are networks of all kinds and sizes from very different areas of society, such as Biology, Physics, Sociology, and many others, going from a couple of nodes to millions, with even more connections. Graphs represent these networks, and there are multiple ways of representing these graphs (see Section 2.1.2 for more information).

Studying graphs and their properties to try to draw conclusions regarding the networks they represent is a science that has been around for a while now. This field of study is called *Network Science* and can be described as "an attempt to understand networks emerging in nature, technology, and society using a unified set of tools and principles" [3]. The evolution of technology in the past decades allowed for new types of studies in this field, where it is now possible to study bigger networks and take advantage of the computational capabilities and accessibility to further develop this field by making these studies available to anyone anywhere in the world.

One way to study this network is through metrics and measures, and based on the interpretation of the values they return, conclusions can be drawn. Centrality measures are one of these available measures to the user, that can use them for multiple purposes. They give each node/edge a value of importance. There are many different centrality measures, some related to nodes and some others related to edges. There are more centrality measures related to nodes than related to edges. Each measure is distinct, attributing this value based on different characteristics of the node/edge in the network. In this work, the values that the centrality measure provides are used for percolation. The process of (inverse) percolation consists in ordering the structures based on the value the centrality measure attributed to every single one of them present in the network, and then based on that order, remove the structures, and study the impact that change causes to the network. Different measures produce different results. Therefore each measure has its pros and cons.

## 1.1 Objectives

A considerable amount of measures have been proposed and continuously are throughout the years. A part of their value comes from being publicly available and easy to use by anyone interested in pursuing this field of study. An example of that is the Spanning Edge Betweenness, an edge centrality measure [4]. For recent measures, such as this one, it is important that they are publicly available, so they can be reviewed by the scientific community. With that in mind, it is important to make the measures available

in the most widely used technologies. Python is now the programming language of choice to study networks, and for a centrality measure to be publicly available and easy to use it should be integrated into a Python library (more details in Section 2.2.2). In this study we also explore the approximation of centrality values, and if it is viable or not for Spanning Edge Betweenness. We also aim to compare Spanning Edge Betweenness with other centrality measures, to better understand its characteristics and contributions, using percolation.

## **1.2 Outline**

In this thesis, in Section 2, basic concepts of Network Science, including graph definitions and representations while also presenting different network models, are introduced. A brief explanation of Spanning Edge Betweenness can be found in Section 3, together with efforts to reduce the computational cost of this centrality measure. In Section 4 is the calculation of the approximation of Spanning Edge Betweenness for weighted graphs, along with results for such implementation. Finally, in Section 5, there is the percolation process, where the results produced by Spanning Edge Betweenness are compared to the ones produced by Edge Betweenness Centrality, another centrality measure also related to edges.

# 2

## Network Science Background

### Contents

---

2.1 Network Science Concepts . . . . .	7
2.2 Centrality Measures and Connectivity . . . . .	12
2.3 Related work . . . . .	18

---





**Figure 2.1:** Circles represent nodes and lines represent edges. Edges in Figure 2.1a are bidirectional, meaning they connect both nodes to one another in both ways, while in Figure 2.1b every edge is unidirectional connecting one node to another, just in that specific way, pointed by the arrow. In red we have the shortest path between nodes *A* and *D*.

## 2.1 Network Science Concepts

In this section, some basic but fundamental Network Science concepts are introduced and briefly explained, to make the rest of the document understandable to the reader.

### 2.1.1 Graph Theory

Graphs are the mathematical representation of networks. These networks can come from all different types of backgrounds, being able to represent the most diverse complex systems' structure and dynamics. A vertex/node represents an entity in a graph. This entity can be a person (social networks), a player (game theory), a computer (IT networks), a neuron (brain networks), among others. An edge/link represents the connection between different entities. These connections can represent physical structures (a pipe), social interactions (between different persons), co-fluctuations (brain activity), among others.

A graph, denoted as  $G = (V, E)$ , is a tuple of sets, where  $V$  is a set of nodes(/vertices) and  $E$  is a set of edges(/links), such that  $E \subseteq V \times V$ . The size of the set  $V$  is denoted by  $N = |V|$  and indicates the number of nodes in the graph. The size of the set  $E$  is denoted by  $L = |E|$  and indicates the number of edges in the graph. Two nodes  $(i, j)$  are adjacent if there is an edge  $e \in E$ , between them. These nodes are called *neighbours*.

A graph is called *undirected* if  $E$  is a set of unordered pairs, meaning that every edge is bidirectional, therefore the edge  $(i, j)$  is equal to the edge  $(j, i)$ . Oppositely, a graph is called *directed* if  $E$  is a set of ordered pairs, so that the edge  $(i, j)$  is different from the edge  $(j, i)$ . A tree is a graph with no cycles. A spanning tree  $T(V, E')$ , where  $E' \subseteq E$ , is a subgraph of  $G$  that is a tree and contains all nodes of  $G$ , with  $L' = |E'| = |V| - 1$ . A minimum spanning tree(MST) is a spanning tree in which  $\sum_{e \in E'} w(e)$  is minimum among all spanning trees of graph  $G$ .

A path is a sequence of  $x$  edges which joins a sequence of  $x + 1$  nodes. A graph  $G$  is said to



**Figure 2.2:** Figure 2.2a represents a directed weighted graph, where a numerical value is attributed to every edge. This values can have multiple different meanings, for instance it could represent the flow of a water system. In Figure 2.2b we can see the shortest path between nodes  $A$  and  $D$ , different from Figure 2.1b because of the addition of edges' weights.

be connected if there is a path between every two distinct nodes. Otherwise, graph  $G$  is said to be disconnected. In disconnected graphs, there are connected components, denoted by  $C$ , where each connected component is a maximal set of nodes,  $C \subset V$ , such that  $C$  forms a connected graph, i.e., there is a path connecting every two distinct nodes in  $C$ .

A weighted graph is a tuple  $G = (V, E, w)$ , where  $V$  and  $E$  are sets of nodes and edges, respectively, and  $w$  is a function that assigns to each edge  $e \in E$  a numerical value. These numerical values can represent different things, based on what the network is about. One of the most intuitive is in a flow network, where each weight represents the capacity of each edge.

In unweighted graphs, the shortest path between two nodes  $(i, j)$  is the path with the smallest size, i.e., the path that contains the least amount of edges  $e \in E$  and connects  $i$  and  $j$ . On the other hand, on a *weighted graph*, the shortest path is not always the path with the least amount of edges, instead, it is the path that minimizes  $\sum_{e \in \sigma(i, j)} w(e)$  where  $e$  is an edge in a path  $\sigma(i, j)$  between nodes  $i$  and  $j$  and  $w(e)$  is the weight of edge  $e$ . In Figure 2.2b we can see an example of the case explained above, where the shortest path between nodes  $A$  and  $D$  is not the most intuitive and direct path between the two nodes, because the edge that is connecting them has a large weight. By comparing figure 2.1b to Figure 2.2b we can see the impact a change from unweighted to the weighted graph can have in the shortest paths of the graph. The distance between two nodes  $d(i, j)$  is the size of the shortest path between nodes  $i$  and  $j$ . If there is no shortest path, then  $d(i, j) = \infty$ . The greatest distance between any two nodes in  $G$  is called the *diameter*, which can be a useful metric.

## 2.1.2 Graph Representations

A graph can be represented in different ways, with different data structures. Often, the option depends on the context and the operations needed to be performed. Next, we present some possible representations:

**Adjacency matrix** of a graph  $G$ , is a matrix  $A = |V| \times |V|$  that contains the information if two nodes  $i$  and  $j$  are adjacent or not. The entries of the matrix  $A_{ij} \in \{0, 1\}$ , are defined by:

$$A_{ij} = \begin{cases} 1, & \text{if } (i, j) \in E \\ 0, & \text{otherwise} \end{cases} \quad (2.1)$$

Spectral graph theory studies the relationship between a graph and the eigenvalues and eigenvectors of its adjacency matrix.

**Incidence matrix** of a graph  $G$  is a matrix  $F = |V| \times |E|$  that encodes the relationship between the nodes and edges of the graph, where each row represents a node and each column represents an edge. The entries of the matrix  $F$  for a *directed weighted* graph are defined by:

$$\begin{cases} F_{ie} = w(e), & \text{if } e = (i, j) \in E \\ F_{je} = -w(e), & \text{if } e = (i, j) \in E \\ F_{ie} = 0, & \text{otherwise} \end{cases} \quad (2.2)$$

It is worth mentioning that for unweighted graphs,  $w(e) = 1$ .

**Edge list** of a graph  $G$  is a  $|E| \times 2$  matrix that is used to represent a graph as a list of its edges. If the graph  $G$  is a weighted graph, a third column should be added to encode the weight of each edge in the list.

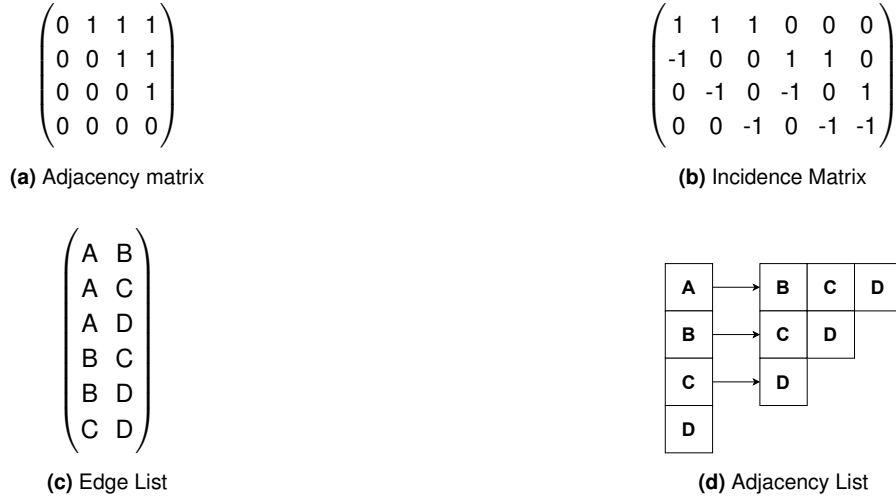
**Adjacency list** of a graph  $G$  is a middle ground between an edge list and an adjacency matrix, where there is a vector of size  $|V|$  representing every node on the graph, and for each node on that vector, there is a list of adjacent nodes.

### 2.1.3 Network Models

Real networks are often analysed and used in studies, however, in the absence of the real structure, it is possible to generate artificial networks based on specific models that try to reproduce the same characteristics as of those networks/systems observed in the real world. In this section we approach some of these models.

#### 2.1.3.A Random Networks

Random Network was the first model proposed to mimic the properties of real networks. In this model, the number of edges and/or nodes is fixed, the way that the nodes connect is random. There are two different models, one by Erdős and Rényi [5]:



**Figure 2.3:** Figures 2.3a, 2.3b, 2.3c and 2.3d are visual examples of the representations presented above for the graph from Figure 2.1b.

- $G(N, L)$  model where  $N$  labelled nodes are connected by  $L$  randomly chosen edges from the set of all possible edges in the graph, regarding the existing nodes.

and other by Gilbert [6]:

- $G(N, p)$  model where  $N$  labeled nodes are connected with probability  $p$ . On the other hand, one could take the complete graph and erase edges with probability  $q = p - 1$ .

We should highlight that the network created by these models can contain isolated nodes, i.e., nodes that are not connected to any other node, but that are still a part of the network. The network is called completely connected if all its nodes are connected, i.e., there are no isolated nodes.

The networks generated by these models are truly random, however with the appearance of real large networks, the characteristics of those did not coincide with the models, therefore they are not adequate to represent real networks.

### 2.1.3.B Small-World Effect

The small-world property, also known as six degrees of separation, became very famous because of a study regarding social networks, by Milgram [7]. According to this study, two individuals in the United States of America, have six or fewer intermediates in their acquaintance chains using the "small world method", introduced by Milgram [8]. This experimentation states that real networks show that people are closer to each other than predicted. Knowing that real networks are not well represented by the random network model, and considering Milgram's findings, Duncan Watts and Steven Strogatz proposed an extension to the random network models previously mentioned [9]:

- Starting from a regular ring lattice with  $N$  nodes, every node has the same degree,  $k$ , and is connected to the closest neighbours, then the procedure of random rewiring is applied to the regular ring lattice, where each edge is rewired at random with probability  $p$ .

Notice that if  $p = 1$  then we have a random network model because every edge gets rewired. This model is closer to real networks regarding the clustering coefficient, however fails to explain the degree distribution.

### 2.1.3.C Scale-free Networks

The increase of computational power led to the possibility of analysing many real large networks. To build a network model that could accurately represent these large networks, the characteristics, specifically degree distribution, should be taken into account. However, when the World Wide Web (WWW) got sampled and mapped out [10], the conclusion reached was that the random network model was not able to generate networks similar to the real large networks. The degree distribution of the WWW was well represented by a Power Law distribution [10], defined as:

$$p_k = k^{-\gamma} \quad (2.3)$$

A network with a Power Law degree distribution, also known as the 80-20 rule, has many small degree nodes – the vast majority of nodes belong to this small degree part of the distribution (correspondent to the 80% in the rule mentioned above) – while very few nodes have a very high degree value – also known as hubs (correspondent to the 20% in the rule mentioned above). Since these hubs can have a great impact on the system's behaviour, this type of network gained an important role in the study of complex systems.

The first *scale-free network model* was created by A.L. Barabasi and R. Albert, the *B-A model* [11]:

- At each time step, the network grows, adding a new node, connecting it to another  $m$  already existent nodes. These connections are probabilistic, giving preference to nodes with a higher degree, making big nodes even bigger, creating hubs. These two processes are called *growth* and *preferential attachment*, two key features of real networks.

Another scale-free model was created by Dorogovtsev-Mendes-Samukhin, called *DMS model* [12]:

- In this model, as in the previous one, every time step a node is added to the network, but instead of connecting itself to the already existent nodes, it randomly chooses an edge and connects to both ends of it. As stated in [12], there is preferential attachment, but unlike the *B-A model*, it comes naturally. The networks generated by this model also achieve a higher cluster coefficient than the ones generated with the *B-A model*.

## 2.2 Centrality Measures and Connectivity

### 2.2.1 Centrality Measures

This section is divided into two subsections. First, we present the centrality measures, and how to compute them, regarding nodes and edges. We should note that not every formula was originally presented with the same notation, however, in the present work, this notation will be made uniform for better understanding.

#### 2.2.1.A Measures related to Nodes

**Degree Centrality** of a node  $i$  is the number of other nodes directly linked to node  $i$ :

$$D(i) = \sum_j A_{ij}, \quad (2.4)$$

where  $A_{ij}$  is the  $ij$ -th element of the adjacency matrix  $A$  of the graph. The degree centrality could be normalized, by dividing the above expression by  $(n - 1)$  for each node, where  $n$  represents the number of nodes in the graph. For directed graphs, the degree centrality is composed by the in-degree centrality and the out-degree centrality, which can be computed similarly.

**Eigenvector Centrality** calculates the centrality for each node based on the centrality of its neighbours [13]. The Eigenvector centrality of a node  $i$  is the sum of the centralities of  $i$ 's neighbours:

$$x(i) = \frac{1}{\lambda} \sum_j A_{ij} x_j, \quad (2.5)$$

where  $A$  is the adjacency matrix of the graph, with  $\lambda$  being the largest eigenvalue of  $A$ . *Alpha centrality* [14], can be seen as a generalization of Eigenvector centrality to directed graphs.

**Katz Centrality** introduced by Leo Katz [15], is similar to the Eigenvector Centrality mentioned above, as the centrality of a node  $i$  depends on the centrality of its neighbours, but also on a constant value:

$$x(i) = \alpha \sum_j A_{ij} x_j + \beta, \quad (2.6)$$

where  $A$  is the adjacency matrix of the graph. Both  $\alpha$  and  $\beta$  are positive constants, and  $\alpha < \frac{1}{\lambda}$ , where  $\lambda$  is the largest eigenvalue of  $A$ .

Katz centrality can be viewed as an improvement to the Eigenvector Centrality previously mentioned, as it solves the problem of having a node with zero centrality, as  $\beta$  is a constant value.

**Closeness Centrality** introduced by Gert Sabidussi [16], as the name suggests, represents how close a node is to all reachable nodes. The higher the value of Closeness of a node the more central that node is:

$$C(i) = \frac{n-1}{\sum_{j=1}^{n-1} d(i, j)}, \quad (2.7)$$

where  $n$  represents the number of nodes and  $d(i, j)$  the shortest-path distance from  $i$  to  $j$ . A measure related to this one, called *global reaching centrality*, can be found in [17].

**Betweenness Centrality** of a node  $v$  is the sum of the fraction of all shortest paths that pass through node  $v$  for every combination of start node and end node on the graph:

$$B(v) = \sum_{i, j \in V} \frac{v \in \sigma(i, j)}{\sigma(i, j)}, \quad (2.8)$$

where  $v \in \sigma(i, j)$  represents the number of shortest-paths between  $(i, j)$  that goes through the node  $v$ , and  $\sigma(i, j)$  represents the number of shortest-paths between  $(i, j)$ . Betweenness centrality measure was originally introduced by Linton C. Freeman [18], however a more recent algorithm was proposed [19].

**Percolation Centrality** is similar to the aforementioned Betweenness centrality, in the sense that it is also calculated based on the paths that go through some node  $v$  regarding all possible paths for each possible pair of start and end nodes. However, these paths, unlike in the Betweenness centrality, are not shortest paths but "percolated paths", meaning that the "source is percolated" [20]. The Percolation centrality for a node  $v$  at timestamp  $t$  can be calculated by:

$$P(v) = \sum_{i \neq v \neq j} \frac{v \in \sigma(i, j)}{\sigma(i, j)} \frac{x_s^t}{[\sum x_i^t] - x_v^t}, \quad (2.9)$$

where  $\frac{x_s^t}{[\sum x_i^t] - x_v^t}$  represents the relative importance of each path originated from the source to the centrality of node  $v$ . For weighted graphs the edge weights must be greater than zero.

**Harmonic Centrality** was presented as an improvement to the Closeness centrality [21], as it allows to have an accurate centrality measure for nodes in a network that has pairs of unreachable nodes:

$$H(i) = \frac{1}{\sum_{d(i, j) < \infty, i \neq j} d(i, j)}, \quad (2.10)$$

where  $d(i, j)$  represents the size of the shortest-path from  $i$  to  $j$ .

**PageRank Centrality** is a measure based on the citation/links on the Web [22], that looks at nodes of the graph as pages on the web. It is used to calculate a page's importance or value. Similarly to the Eigenvector centrality measure, the PageRank centrality of a page also depends on other pages that point towards this page, not only on their PageRank centrality value but also on the number of their outgoing links. PageRank centrality of a node represents the fraction of time a random walker would spend on this node. The walker follows out-edges with probability proportional to their weight. In this context of following links going from page to page, it is possible that a user stops following links and stops on a page, the probability of that happening in each step is the damping factor  $d$ . The PageRank centrality value for each node can be calculated with:

$$PR(i) = (1 - d) + d \left( \frac{PR(P_1)}{L(P_1)} + \dots + \frac{PR(P_n)}{L(P_n)} \right), \quad (2.11)$$

where  $P_1 \dots P_n$  are all different pages, pointing/linking to page  $i$  and  $L(P_1) \dots L(P_n)$  is the number of different links going out of page  $1 \dots n$ , respectively.

### 2.2.1.B Measures related to Edges

**Edge Betweenness Centrality** is very similar to the Betweenness centrality, but in this case instead of nodes we are focused on the edges. The Edge Betweenness of an edge  $e$  is the sum of the fraction of all shortest paths that pass through  $e$  for every pair of start and end nodes on the graph:

$$E(e) = \sum_{i,j \in V} \frac{e \in \sigma(i,j)}{\sigma(i,j)}, \quad (2.12)$$

where  $e \in \sigma(i,j)$  represents the number of shortest-paths between  $(i,j)$  that goes through the edge  $e$ , and  $\sigma(i,j)$  represents the number of shortest-paths between  $(i,j)$ . It was first introduced in [23].

**Degree Product Centrality** The Degree Product centrality of an edge is based on the degree of its adjacent nodes. This measure is used for assortative graphs, however, can be misleading in disassortative graphs as nodes are connected to other nodes with very different degree values. It can be calculated as:

$$Dp(e) = D(i)D(j) \quad (2.13)$$

where  $D(i)$  represents the degree of node  $i$ , and edge  $e$  is the edge that connects nodes  $i$  and  $j$ . A concrete literature reference for the definition of this measure was not found, however as it is exclusively based on the degree of the nodes, which is a concept from graph theory, it is natural to assume that this measure is also from graph theory. This measure, and variants, have been widely used throughout the

years [24–26].

**Bridgeness Centrality** looks onto sizes of cliques, more specifically the cliques' size of the two adjacent nodes of an edge, and also to the clique size that the edge belongs to. It can be calculated by:

$$B(e) = \frac{\sqrt{S_i S_j}}{S_e}, \quad (2.14)$$

where  $i$  and  $j$  are the two nodes connected by  $e$ , and  $S_i$  represents the size of the clique that contains  $i$ .

**K-Path Edge centrality** in contrast to the edge Betweenness centrality mentioned earlier, the K-Path Edge centrality measure does not look into the shortest paths between two nodes that cross an edge, instead, it looks into the  $k$ -sized paths from a node that cross an edge. The centrality value for each node can be calculated by:

$$K(e) = \sum_{i \in V} \frac{e \in \sigma_k(i)}{\sigma_k(i)}, \quad (2.15)$$

where  $\sigma_k(i)$  represents a path with length  $k$  from node  $i$ . The notion of K-Path centrality was introduced in [27].

**Spanning Edge Betweenness** unlike other measures already mentioned, the Spanning Edge Betweenness centrality measure [4], does not take into account the shortest paths of a graph. It looks into which links are fundamental to keep the network connected and which are not. To do so, it takes into account the Minimum Spanning Trees (MSTs) of the graph, and the Spanning Edge Betweenness of an edge is the fraction of minimum spanning trees that edge is a part of, and can be calculated by:

$$S(e) = \frac{e \in \tau_G}{\tau_G} \quad (2.16)$$

where  $e \in \tau_G$  represents the number of MSTs that contain edge  $e$ , and  $\tau_G$  represents the total number of MST of graph  $G$ . This measure is bound to be between 0 and 1. For unweighted graphs, a value of 1 means that if the edge is removed the network breaks apart.

In table 2.1 there is complementary information regarding each metric presented above.

**Table 2.1:** In this table, there is more information regarding each centrality measure previously presented. **Edges/Nodes, Directed/Undirected** and **Weighted/Unweighted** columns represent if a measure is exclusive to any of the categories or not. **Complexity** represents the asymptotic worst case of calculating the measure to every structure in the network. **Definition** is a reference to where the measure was defined/presented. **Comparisons** references the papers where measures present in this table were compared, mainly regarding percolation. In those papers, there may be measures that are not in this table/work. **Libraries** represents if the measure is present on NetworkX (NX), iGraph, or in both or neither one of these Python libraries

Centrality Measure	Nodes /Edges	Directed /Undirected	Weighted /Unweighted	Time Complexity	Definition	Comparisons	Libraries
Degree	Nodes	Both	Both	$O( V ^2)$	[28]	[1, 29]	Both
Eigenvector	Nodes	Both	Both	$O( V ^3)$	[13]	[1]	Both
Katz	Nodes	Both	Both	$O( V ^3)$	[15]	[1, 21]	NX
Closeness	Nodes	Both	Both	$O( V  E )$	[16]	[1, 21]	Both
Betweenness	Nodes	Both	Both	$O( V  E  +  V ^2 \log  V )$	[18]	[1, 20, 21, 29]	Both
Percolation	Nodes	Both	Both	$O( V ^3)$	[20]	[20]	NX
Edge Betweenness	Edges	Both	Both	$O( V  E  +  V ^2 \log  V )$	[23]	[30, 31]	Both
PageRank	Nodes	Both	Both	$O( V  +  E )$	[22]	[1, 21]	Both
Harmonic	Nodes	Both	Both	$O( V  E )$	[32]	[21]	Both
Degree Product	Edges	Both	Both	not found	not found	[2, 31]	Neither
Bridgeness	Edges	Both	Both	$O( V  E )$	[31]	[2, 31]	Neither
K-Path edge	Edges	Both	Both	$O(k^3  V ^{2.5} \log  V )$	[27]	[2]	Neither
Spanning Edge Betweenness	Edges	Undirected	Both	$O( V ^{2.5})$	[4]	[30]	Neither

### 2.2.2 Libraries

NetworkX (<https://networkx.org>) [33], is a Python package oriented to networks, as the name suggests. It is useful for the creation and manipulation of the structure, dynamics, and functions of complex networks. Not only have many of the centrality measures that were referred to in this paper implemented, but it also has generators for graphs based on the best-known models, high tolerance for different types of nodes, and both nodes and edges can hold arbitrary extra data. It is constantly updated, making it as up-to-date as possible. This flexibility of NetworkX allows for the representation of different types of networks, while also, as a Python package, being very attractive for all kinds of users – not only for the ones more familiar with programming but also for those with all sorts of different backgrounds. NetworkX is also able to perform numerical linear algebra and drawing by making use of multiple Python packages.

Another very famous network tool is igraph (<https://igraph.org>). Unlike NetworkX, igraph is not only a Python package, it can also be used in other programming languages such as R, Mathematica and C/C++. It focuses on efficiency, portability and ease of use. The fact that it provides APIs for different languages allows for people of different programming backgrounds to be at ease with the preferred language, besides choosing the most optimal implementations. igraph and NetworkX should not be seen as competitors but rather as complements, each one having unique functions implemented when compared to each other. Just like NetworkX, igraph is also updated very regularly.

Python (<https://www.python.org>) is a very well known programming language. Its easiness and accessible characteristics are what makes it so good and widely used. To start it is very easy to learn, it can easily be an entry language into the programming world, or if that is not the goal, one can stick to it and still do almost anything, making it a very powerful programming language. It runs everywhere and it is very easy to set up, which is a big plus for everyone that uses it. The fact that it is open-source is a plus for someone that wants to dig deeper into this programming language.

Taking into consideration everything that was just stated, about the importance of those two libraries on making working with networks much easier. Also, the fact that Python is such a good programming language for this same subject, not just because it is very powerful but also because it is easy to use. One can easily understand the importance of a centrality measure being publicly available to everyone in one of those libraries that take advantage of a language such as Python.

### 2.2.3 Robustness

Networks represent real-life entities and connections. Therefore networks are susceptible to *events*, as the real world is continuously changing. A network that can withstand these events is said to be *robust*. These events can be represented by the addition or deletion of network structures, such as nodes or edges. In a citation network, every time a new paper is published there are new edges and a new node

that have to be added to the network. In a water supply network if a pipe is broken an edge has to be deleted from the network.

The study of robustness can be done with two different perspectives – increasing the resilience of a network or fragmenting the connectivity of a network. Here the questions are: for a given graph  $G$ , which modifications in its structure (adding or removing elements such as nodes and/or edges), possibly restricted by a budget or other constraints, resulting in the highest increase of robustness, or lead to the most broken graph? The type of events in which elements are added to the graph is often more related to the first perspective, while on the other hand, the type of events that delete elements from the graph is more related to the second. There are multiple different available measures/methods to assess the robustness of a network. Naturally, each one of these serves different background areas, with different purposes.

The deletion events, also known as *faults*, can either be random failures or targeted attacks. As the name suggests, a random failure can affect any edge or any node that belongs to a network. On the other hand, in a targeted attack, the attacker relies on a certain strategy to pick the most important elements to cause the most damage to the network. The modeling of repeated deletions and their effects is known as *percolation* [34].

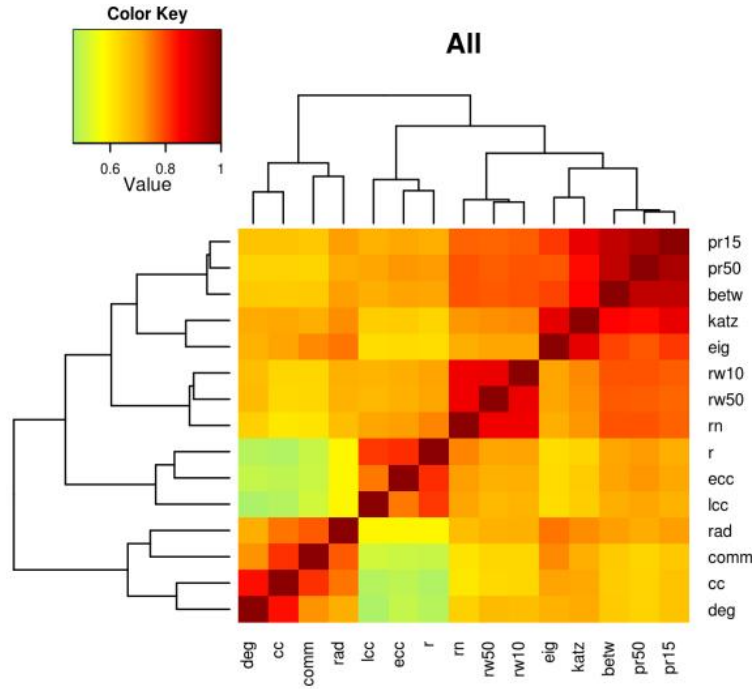
One way of attributing values to network elements relative to their importance is by using centrality measures. Centrality measures attribute a numerical value to each node or edge, that allows for them to be ordered. There are some centrality measures specific for nodes and others specific for edges. The choice between focusing on the nodes or the edges, and corresponding measures, depends on the type of the network and on the context of the event. In the next section, some measures are reviewed.

Since there are many definitions for robustness, in this work we define robustness as the capability of the network to remain connected, or be highly connected, after *events* occur. The size of the GCC (giant connected component), number of Connected Components, a fraction of nodes/edges belonging to the GCC, normalized susceptibility, among others, are available measures to assess such capability of a network, as percolation is performed on the network.

## 2.3 Related work

### 2.3.1 Networks

When researchers first started using networks in their studies the computational power did not allow for real-world large networks to be studied. Therefore they had to either use networks created based on models (briefly explained in Section 2.1.3) or small real-world networks. In both [29,35] the authors studied and compared the robustness of the different network models, when confronted with both random failures and targeted attacks, and reached interesting conclusions where some models are more robust



**Figure 2.4:** Image retrieved from [1], illustrating correlation between different studied centrality measures. A color scale was used, where green represents less similarity and red represents more similarity. The three similarity clusters are: betw-pr15, katz-eig and deg-cc.

to a type of *event* and others perform the other way around. However as the years went by the computational power increased a lot and for some time now it is possible to work with very large real-world networks, like the ones used in [30], where the largest has millions of nodes.

Even with this huge growth in computational power, followed by the growth in the networks' size there are measures that are still unviable to calculate, by being too computationally expensive. However it would be very clever to study similarities between centrality measures, where one centrality measure could be closely approximated by another one much cheaper, that would be viable to be calculated. A study in this area was performed [1], reaching interesting results, by being able to create three clusters of similar centrality measures, where the cheaper measures can be used instead of the similar but more expensive ones (see Figure 2.4 for more information).

### 2.3.2 Nodes vs Edges

As stated before, every study in this area considers some type of changes or perturbations inflicted on the network. The name given to these occurrences in this work is *events*. The adequate *events* for a network vary with the type of network and the context/area of the study to be performed. As an example, in a contacts network where the nodes represent humans and edges represent face-to-

face interactions, and the objective is to study epidemics, it does not make sense to remove nodes, as humans cannot simply disappear, instead it makes sense to remove edges, by reducing face-to-face interactions between humans, represented by the nodes. It is important then to categorize papers, in those that target nodes and those that target edges. In [30, 36–38] edges are the target of the events. On the other hand in [1, 29, 35, 39] nodes are the target.

### 2.3.3 Approximating Centrality Measures

Centrality measures are widely used in *Network Science*. This fact, together with the continuous growing size of the networks studied, leads to the necessity that these centrality measures are computationally cheap to calculate. The best way of achieving such goal is, instead of calculating the exact measure, algorithms to calculate approximations are created.

An example of this can be seen in [40, 41], where based on a randomized approximation algorithm the centrality measure values are calculated for the given networks. Another example of a similar method can be found in [42], where the algorithm is based on "an adaptive sampling technique" which allows for better computational times. The multiple different proposed algorithms have their pros and cons, i.e., there are differences between each proposed algorithm, and the authors always argue why their method is better from the ones already proposed. For example, in [43], the authors state "...we also get good approximations for the betweenness of unimportant nodes" as an advantage over the methods that were already introduced at that time.

With the evolution in technology, new methods for approximating centrality measures' values were developed. An example of this is the use of machine learning. By using neural networks, the authors state that they can achieve results that are very close to the exact ones only using a fraction of the time [44, 45].

### 2.3.4 Robustness

Robustness is a well-known topic in the network area of interest. It is not a recent subject of study, in fact the first works date to more than 50 years ago [46]. This first work was in the area of network reliability, taking into consideration connectivity measures. Since then this topic has received much attention from researchers from all kinds of different fields. These can be so heterogeneous that some look to study robustness as a way of increasing the capabilities and resilience of a network (telecommunications, power grid), and some others are interested in this same subject with the intent of breaking the network apart (epidemics). There are multiple different definitions for robustness, some examples are: "fault tolerance against random failures of network components" [47], "we define a robust network as one where the total throughput degrades gracefully under node and link removal" [48], "we interpret network

robustness as a measure of the network's response to perturbations or challenges [...] imposed on the network" [49] and "Robustness is the ability of a network to continue performing well when it is subject to failures or attacks" [36].

As the differences seem obvious there is also something in common to all these definitions provided by all these different authors. By studying what happens to the network when it faces some events (addition or deletion of elements) that alter the network, and based on the network's behaviour when affected by those events, the authors are able to argue if the network is considered robust or not.

The contribution from [47] on robustness is very valuable. In this paper, the authors focused on reviewing robustness. It groups measures, that attribute numerical values to the network in order to address robustness, by "graph-theoretical concepts" such as components, paths, degrees, among others. Being this work a review, those measures are not proposed there. The authors present a table with the name of the measures and the references to where they were originally proposed, along with the year, the name on the available repository and the asymptotic worst-case complexity. Another analysis worth mentioning in this work is a comparison of cited work in terms of the type of disturbance (failure or attack, isolated or sequential), the structure of the network that is affected (nodes or edges) and in which of the groups mentioned above the impact is quantified. Additionally, the authors analysed the Pearson correlations among the measures. Together with runtime analysis for the measures for the same networks, it suggests that is possible to replace an expensive measure with a very similar and much cheaper to compute measure, a possibility that is increasingly under study. However, further theoretical analysis is necessary to assure the similarities obtained practically between measures. It is important to note that some measures have a prohibitive runtime and others fail to differentiate graphs with very different structures. Still, this study differs from my work proposal because percolation is not applied to the used networks, instead these networks are used with the intention of comparing the robustness of the underlying models that were used to create them.

As stated above there are many different measures available to assess robustness, from the most different theoretical concepts, however, regarding the definition of a network as being robust, researchers have not yet, and maybe will never, reach a consensus on how to correctly assess that. As the first works on this subject, also [50] defends that *connectivity* is the most appropriate measure for modelling the robustness of networks, an idea that is also present in [35], by targeting the nodes with higher *connectivity*.

In [2], to assess edge significance on maintaining global connectivity, edge percolation was performed, making use of seven different centrality measures, one of which, *link entropy*, was introduced in this work. Two indices are used to capture critical points: the fraction of nodes belonging to the largest connected component and *normalized susceptibility*. Regarding this last index, often a peak can be observed, it corresponds to the point the network disintegrates, i.e., the threshold of edge percolation

(see Figure 2.5 for more details).

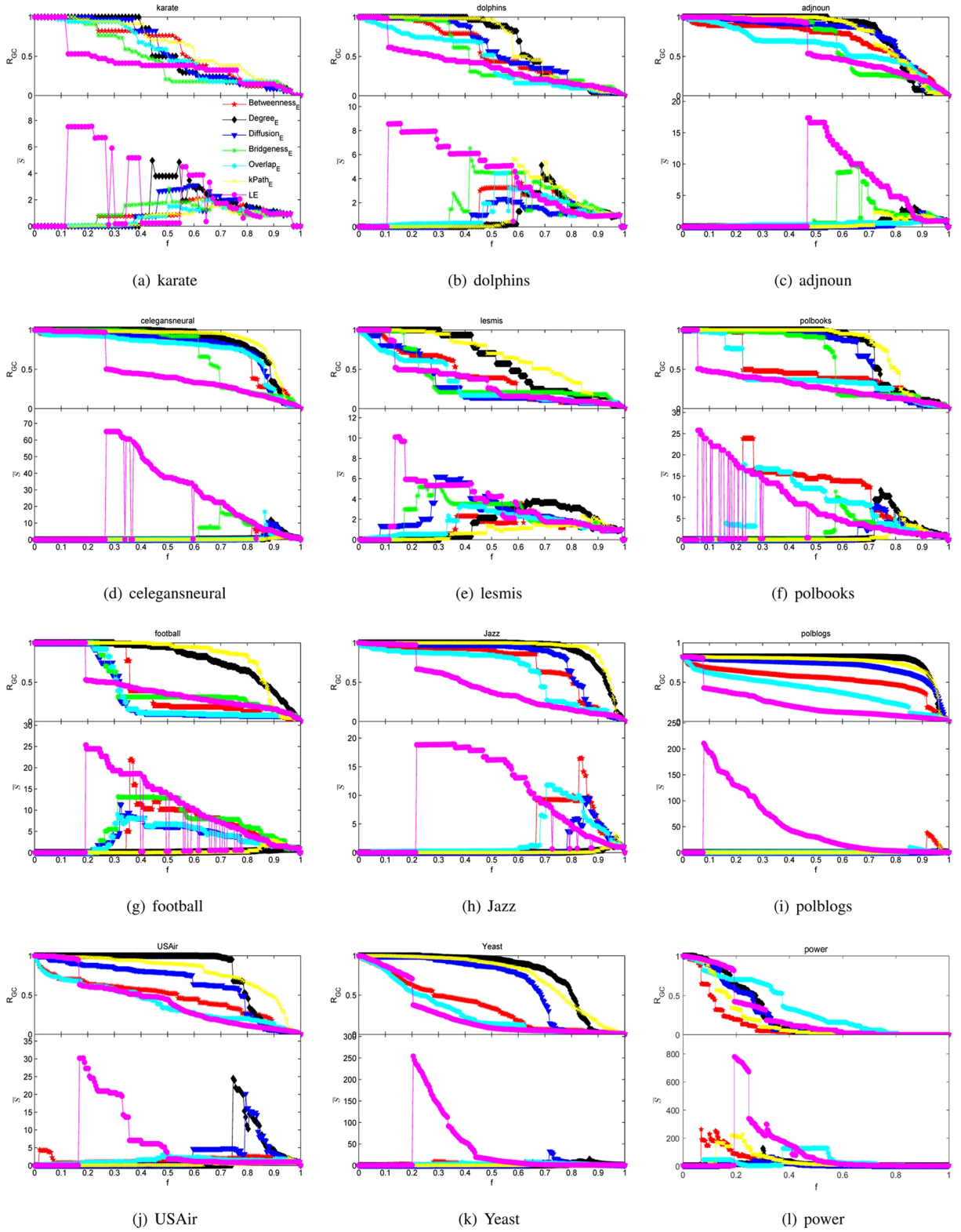
In social networks, it is usual to have connections between similar individuals, however there are some connections between less similar individuals, that are weaker. "The weak ties phenomenon refers to the fact that these connections may play crucial roles in maintaining the global connectivity and holding key functions, such as the flow of new information and knowledge" [31]. This work focus on *Bridgeness centrality*, which is inversely proportional to similarity. The same indices are used in this work to assess robustness: the fraction of nodes belonging to the largest connected component and *normalized susceptibility*. *Bridgeness* is compared to another three centrality measures and performs better in characterising edge significance, according to the authors and the indices chosen. The authors recognize that more large-scale networks should be used in future works [2, 31].

A different approach is used in [32], in which to evaluate the behaviour of the network after a set of events it is used *global efficiency*, a measure that was introduced in [51], as a measure of how efficiently a network exchanges information. A different perspective is given in [39] where *k-robustness* is defined as a measure that reflects the minimal fraction of residual edges after removing *k* nodes. This paper is different to the previously mentioned in the sense that it takes the network attacker's perspective, where it aims to disconnect the network, by trying to remove as many edges as possible when targeting a fixed number of nodes. Another perspective is presented in [48], where the authors propose a metric, called *elasticity*, defined as a relation between throughput and nodes remaining in the network after *events* occur. Another measure was proposed in [49], called *R-value*, that reflects the robustness of the network. Curiously this measure is not defined as a single measure, instead it is defined as a weighted linear norm of a set of measures or a constrained model, where some or all the measures belonging to the set can be upper and/or lower bounded. The goal is to pick measures that are independent of one another to best reflect the different characteristics of a network.

Also on the topic of network robustness, a different approach is given in [36], where the criteria for a measure being useful to assess network robustness is that it needs to reflect the addition of edges to the network and take into account alternative paths, where it states that "the only measure that gives the desired evaluation for the example graphs and also measures back-up paths in the graph, is the effective graph resistance", that was proposed by the same author [52]. As stated in [37, 38] *effective resistance* is related to *Spanning Edge Betweenness*, proposed in [4]. Different ways of calculating *Spanning Edge Betweenness* are proposed with the objective to make this measure viable to be used in real-world networks, as the exact value is stated to be too costly to calculate [30, 53]. Both works focus on an approximation as it is cheaper to compute. It is important to state that these studies differ from my work proposal because I aim to implement the exact measure. An interesting study was made [53], where a fraction of the edges/vertices with higher centrality numerical value were protected and only the edges/vertices that were not protected could be removed. Then the authors examined the probability

that the network remained connected. The authors also applied node percolation by deleting the most important nodes, where the node's importance was calculated using different centrality measures, to a network and measured the number of connected components. The conclusion that can be reached is that the measure that creates more connected components is better at identifying the critical nodes. In [30] the authors compared the noise resilience of different centrality measures when the network was subject to addition or deletion of edges, which follow three different strategies. The resilience was measured by comparing the evolution of two distinct measures: *average relative change* and *Jaccard similarity*.

Epidemics simulation is one area where these two metrics are used [37, 38], where the objective is to sparsify the network to decrease the computation cost while maintaining the network's characteristics by removing edges that are not important for disease spreading. In both works, the measure was compared against other methods of sparsification with multiple evaluation criteria and performed better in every category, allowing the authors to argue that it is a good measure to perform sparsification based on.



**Figure 2.5:** Image retrieved from [2], illustrating edge percolation on twelve different networks with seven different structural measures. Each plot has two parts, the top one for the fraction of nodes belonging to the largest component and the bottom one for the normalized susceptibility.

# 3

## Spanning Edge Betweenness – Implementation and Parallelism

### Contents

---

3.1 Theory . . . . .	27
3.2 Implementation . . . . .	28
3.3 Results . . . . .	31

---



In this Chapter there is a brief introduction to the theory behind Spanning Edge Betweenness centrality. After that, we will take a closer look into the implementation of the aforementioned measure, where there were proposed a couple of methods to improve the calculation time of said measure, and finally compare the results for the methods tested. The code referenced in this chapter is published in an online repository<sup>1</sup>.

## 3.1 Theory

SEB is a centrality measure that calculates how strongly connected a network is, i.e., which links are fundamental to keep the network connected and which are redundant. This metric can be defined as the fraction of Minimum Spanning Trees (MSTs) where a given edge is present. This metric was initially developed to be used in phylogenetic trees [4], but quickly other applications started to emerge, one of them being the study of network robustness.

SEB can be formally defined as:

$$SEB(e) = \frac{e \in \tau_G}{\tau_G} \quad (3.1)$$

where  $G = (V, E)$  is a connected, undirected and weighted graph,  $V$  is the set of nodes and  $E \subset V \times V$  is the set of edges of the graph, and  $e \in E$ .  $e \in \tau_G$  represents the number of MSTs where edge  $e$  occurs, and  $\tau_G$  is the total number of MSTs of the graph. To count how many MSTs exist in  $G$ , and in how many  $e$  is present we can rely on Kirchhoff's matrix tree theorem [54] for unweighted networks and on Eppstein [55] for weighted networks.

In [4] is explained how to compute both  $e \in \tau_G$  and  $\tau_G$ , by using both aforementioned theorems. Computations can however be resumed to calculating determinants of matrices, as these take most of the computation time and are predominant throughout the implementation.

### 3.1.1 Determinants

The determinant can be seen as a scalar value that is a function of the entries of a square matrix. A minor of a square matrix is the determinant of a smaller square matrix obtained from the first by deleting one or more rows or columns. There are multiple ways to calculate a determinant directly, such as using the *Leibniz's formula for determinants*, based on permutations of matrix elements or using the *Laplace expansion*, based on minors. However, these methods are inefficient, therefore new methods of calculating determinants have been developed.

---

<sup>1</sup><https://github.com/ASTeixeira/SpanningEdgeBetweenness>

These are called decomposition methods, and they calculate the determinant of a given matrix by writing the matrix as a product of matrices whose determinants can be more easily computed. Examples are the *LU decomposition* or the *QR decomposition*.

The *QR decomposition* decomposes a matrix into a product of an orthogonal matrix and an upper triangular matrix. This decomposition was published in 1961 by both John G.F. Francis [56] and Vera N. Kublanovskaya [57], independently. The *LU decomposition* was introduced by Tadeusz Banachiewicz in 1938 [58]. This decomposition factors a matrix as the product of a lower triangular matrix and an upper triangular matrix. This decomposition is particularly useful because the determinant of a triangular matrix is the product of the elements of its diagonal. This property is used throughout the implementation.

## 3.2 Implementation

In this section we look into the implementation of SEB. We chose to implement SEB using the Python programming language, specially because we aimed to integrate this measure in NetworkX, a Python package.

In Algorithm 3.1 there is pseudocode regarding the SEB calculation for weighted networks. One must start by ordering the edges by their weights, in increasing order. Then, for each connected component formed by edges with the same weight, calculate SEB for each edge in each connected component. After that, contract all edges such that each connected component becomes a single vertex. Repeat until all edges are processed.

---

### Algorithm 3.1: Weighted Network

---

```

1 begin
2    $edges \leftarrow \text{sortEdgesByWeight}(G.edges())$ 
3    $Laplacian \leftarrow \text{laplacianMatrix}(G)$ 
4    $LaplacianToKirchhoff \leftarrow \text{RemoveRowColumn}(Laplacian, 0, 1)$ 
5    $TotalMSTs \leftarrow \text{CalculateDeterminant}(LaplacianToKirchhoff)$ 
6    $n \leftarrow 0$ 
7   while True do
8     if StillInTheSameWeightValue() then
9        $\text{AddEdgeToLaplacian}(edges[n])$ 
10       $n \leftarrow n + 1$ 
11     if AnyEdgeProcessed() then
12        $\text{BuildSCCs}()$ 
13        $\text{CalculateSEBEachSCC}()$ 
14       if NoMoreEdges() then
15          $\text{break}$ 
16        $\text{ContractEdgesEachSCC}()$ 
17     else
18        $\text{IncreaseWeightValue}()$ 

```

---

The calculation for SEB values of an unweighted network can be seen as a specific case of a weighted network, where all the edges have the same weight, therefore having only one connected component, so the calculation is more direct. Pseudocode for such cases is available in Algorithm 3.2.

---

**Algorithm 3.2:** Unweighted Network

---

```

1 begin
2   Laplacian  $\leftarrow$  laplacianMatrix(G)
3   LaplacianToKirchhoff  $\leftarrow$  RemoveRowColumn(Laplacian, 0, 1)
4   TotalMSTs  $\leftarrow$  CalculateDeterminant(LaplacianToKirchhoff)
5   for edge in G.edges() do
6     SEB  $\leftarrow$  CalculateSEB(edge, Laplacian)
7     updateEdgeSEB(G, edge, SEB)

```

---

As stated before, conventional methods of calculating the determinant of a matrix are not optimal, because they are very inefficient. It is reasonable to expect that packages for scientific computing for Python, such as *SciPy*<sup>2</sup> [59] and *NumPy*<sup>3</sup> [60] use more efficient options for calculating determinants, such as decompositions. With that in mind, a possible solution for the problem of calculating determinants would be to use built-in functions from those packages, however when calculating the determinant of a matrix of a relevant size, built-in functions may cause overflow, so a solution to this problem would be to use built-in functions from the same packages to factorize the initial matrix, and then perform the calculations using logarithmic properties, which allow to represent bigger numbers easier, mitigating the risk of overflow. The decomposition that best fits this use case is the *LU decomposition*, as it possibilites the easiest method to calculate the determinant of a matrix, hence it is the one that will be used. Pseudocode exemplifying these operations is available in Algorithm 3.3.

---

**Algorithm 3.3:** Calculate Determinant

---

```

1 begin
2   LU  $\leftarrow$  LUFactorization(Laplacian)
3   Determinant  $\leftarrow$  0
4   for entry in Diagonal(LU) do
5     Determinant  $\leftarrow$  Determinant + Log10(Absolute(entry))

```

---

The calculation of the exact SEB value for every edge can be heavy and expensive. With that in mind, there is a possible preprocessing that would make the computations faster. As stated before, the SEB value of an edge is the fraction between the number of MSTs that such edge is present and the total number of MSTs of the graph. A bridge on a network is by definition an edge whose deletion increases the graph's number of connected components, therefore this edge has to be present in every MST of the network, and so it is possible to set the SEB value of such edges *a priori*, and temporarily

---

<sup>2</sup><https://scipy.org/>

<sup>3</sup><https://numpy.org/>

remove them from the network, creating subgraphs, which are smaller than the initial graph, making the Laplacian matrix for each subgraph smaller and making the decomposition faster, so the calculation for the remaining edges is lighter and faster. Pseudocode in Algorithm 3.4.

---

**Algorithm 3.4: Unweighted Preprocessing**

---

```

1 begin
2   Bridges  $\leftarrow$  GetBridges(G)
3   RemoveEdgesFromGraph(bridges, G)
4   SCCs  $\leftarrow$  GetSCCsFromGraph(G)
5   for SCC in SCCs do
6      $\lfloor$  CalculateSEB(SCC)
7   SetEdgeAttribute(G, Bridges, SEB = 1)

```

---

The same preprocessing does also apply to weighted networks, because no matter the weight of a bridge it will always be in every MST. The only difference between this preprocessing and the one stated above is that for a weighted network it is crucial to also keep the weight of the bridges, so when they are being added back to the graph, no information is lost. Pseudocode in Algorithm 3.5.

---

**Algorithm 3.5: Weighted Preprocessing**

---

```

1 begin
2   Bridges  $\leftarrow$  GetBridges(G)
3   Weights  $\leftarrow$  GetWeightsFromEdges(G, Bridges)
4   RemoveEdgesFromGraph(bridges, G)
5   SCCs  $\leftarrow$  GetSCCsFromGraph(G)
6   for SCC in SCCs do
7      $\lfloor$  CalculateSEB(SCC)
8   SetEdgeAttribute(G, Bridges, weight = Weights, SEB = 1)

```

---

Another attempt at making SEB faster to calculate led to the use of *PySpark*. *PySpark* is a Python interface for Apache Spark. It allows to easily use the available cores of a machine, with little changes to the implementation.

In the implementation of SEB, the calculation of the value for each edge is independent, therefore it is possible to easily parallelize the calculation. Taking advantage of the capacities of the *PySpark* interface it is possible, with minimal changes to the code, to implement such improvement.

Relying on *PySpark*'s Resilient Distributed Dataset(RDD) and the *MapReduce* programming model the modifications above can be implemented, as seen in Algorithm 3.6.

---

**Algorithm 3.6:** PySpark SEB calculation

---

```
1 begin
2   Laplacian  $\leftarrow$  laplacianMatrix(G)
3   LaplacianToKirchhoff  $\leftarrow$  RemoveRowColumn(Laplacian, 0, 1)
4   TotalMSTs  $\leftarrow$  CalculateDeterminant(LaplacianToKirchhoff)
5   EdgeList  $\leftarrow$  parallelize(G.edges())
6   SEB  $\leftarrow$  EdgeList.map(CalculateSEB(edge, Laplacian)).collect()
7   updateEdgeSEB(G, SEB)
```

---

### 3.3 Results

In this section we look at the results of the decisions made for the implementation, where the aim is to minimize the calculation time of SEB for a network. The machine used to obtain these results has the following specs: Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz, 24 cores and 64GB of RAM.

The following list contains the name and a small description of the networks, real world networks and generated, used to perform tests and obtain results.

- **PowerGrid**(Infrastructure) - A representation of the Western States Power Grid of the United States, compiled by Duncan Watts and Steven Strogatz.
- **Facebook**(Social) - A social friendship network extracted from Facebook, where nodes represent people and edges represent friendship ties.
- **Dolphins**(Animal Social) - A social network of bottlenose dolphins where links represent frequent associations.
- **C.elegans**(Biological) - In this network nodes represent substrates and edges represent metabolic reactions.
- **NetScience**(Collaboration) - A co-authorship in network theory and experiments, where nodes represent researchers and edges co-authorship.
- **Web**(web) - In this network nodes represent web-pages and edges represent hyperlinks between web-pages.
- **ER**(Random) - A random network generated from the *Erdős-Rényi* model.
- **NWS**(Small-World) - A small-world network generated from the *Newman-Watts-Strogatz* model.
- **BA**(Scale-Free) - A scale-free network generated from the *Barabasi-Albert* model.

**Table 3.1:** Real world networks' characteristics

Network	# Nodes	# Edges	Clustering Coefficient	Assortativity	Average Degree
PowerGrid	4941	6594	0.080	0.003	2
Facebook	2888	2981	0.027	-0.668	2
Dolphins	62	159	0.258	-0.043	5
C.elegans	453	2025	0.646	-0.225	8
NetScience	379	914	0.741	-0.081	4
Web	3031	6474	0.564	-0.168	4

In Table 3.1 there are some important characteristics from each real world network, such as number of nodes and edges, clustering coefficient, assortativity and average degree. By looking at these characteristics it is possible to acknowledge that the networks are quite different from each other.

As stated before, conventional methods of calculating the determinant of a matrix are too inefficient, therefore a better approach is to use decompositions, that decompose the original matrix in multiple sub-matrices allowing for a faster calculation. It is reasonable to assume that built-in functions would use these or even better methods for determinant calculation. However when using those built-in functions, they may overflow, when applied to the bigger networks used. The solution is then to use the same packages, but to use the decomposition built-in functions instead. Also stated before was the use of the *LU decomposition* to calculate the determinant of a matrix. This decomposition is available and easy to use in *SciPy*.

*SciPy* is a Python package for scientific computing that has its own group of algorithms and functions regarding linear algebra, but in addition to that, it also has a module that contains low-level functions from the LAPACK library, that stands for Linear Algebra PACKage, originally written in the programming language Fortran 90. The time comparisons between the two different modules can be seen in Table 3.2.

**Table 3.2:** Time that different decompositions take to calculate SEB for each network, in seconds. "scipy's decomposition" stands for the native linear algebra function, and "lapack's decomposition" represents the function from the LAPACK module inside Scipy.

Network	lapack's decomposition	scipy's decomposition
PowerGrid	7936	9182
Facebook	1502	1741
Dolphins	0.60	0.60
C.elegans	307	316
NetScience	78	78
Web	3696	4227

In Table 3.2, both decomposition functions are being compared. When looking at the smaller net-

works, there are no time differences, which could lead to conclude that there is no difference between using one or another, however when looking to the biggest networks, such as PowerGrid or Web there is a considerable difference between functions, with LAPACK's being the best option. It is then possible to conclude that LAPACK's function is faster than Scipy's, justifying our choice of use for the remainder of this work.

As previously stated, a potential improvement to the time duration to calculate SEB for a network would be to find bridges *a priori*, so that edges no longer need to be processed, as the SEB value is known for such cases and also to break a bigger graph into smaller subgraphs, so their Laplacian matrices are also smaller and therefore faster to decompose. Related to this matter, two main questions can be made: does preprocessing improve the calculation time when there are bridges present in the network? and does the preprocessing worsens the calculation time when there are no bridges on the network? The results of this preprocessing for the real world networks is presented in Table 3.3.

**Table 3.3:** Real world networks SEB calculation time without and with preprocessing applied, time in seconds.

Network	# Edges	# bridges	Without preprocessing	With preprocessing
PowerGrid	6594	1611	7936	3126
Facebook	2981	2796	1502	0.8
Dolphins	159	9	0.60	0.35
C.elegans	2025	8	307	212
NetScience	914	30	78	63
Web	6474	512	3696	1134

By looking at Table 3.3 there are some conclusions that can be drawn. We can in fact answer to the first question we stated, the preprocessing is worth with the presence of bridges in the network. This holds true both for networks with a small portion of bridges and networks with a high portion of bridges.

Let us take as an example the C.elegans network for the small portion of bridges. This network has 2025 edges and out of those only 8 are bridges, and the calculation time goes down almost a third of the original time, which shows that even for a network with a percentage of edges being bridges so low it does still pays off to apply the preprocessing.

An example on the other side of the spectrum, i.e., a network with a high percentage of bridges is the Facebook network, with 2796 bridges out of 2981 edges and the effect on the calculation time is tremendous with the time going from 1502 seconds to 0.8 seconds, which leaves no doubts about the value of preprocessing when the network has a significant amount of bridges.

After analyzing the results of the real world networks it is also important to analyze and draw conclusions from generated networks, based on well-known models. In Table 3.4 the same comparison is made for networks generated from the Newman-Watts-Strogatz model, considering different sets of

parameters.

By looking at the mentioned table it is obvious that there is not a single generated network with a bridge. So now it is not possible to draw any conclusions regarding the first question, but it is possible to address the second, does the calculation time get a lot worst when no bridges are present? By looking at the values present in the two most rightward columns, it is expected that without preprocessing the time is better, because there is no advantage in preprocessing such networks and time is spent in this process. However, it is also possible to see that the calculation time does not increase significantly, which allows us to conclude that with networks generated from the Newman-Watts-Strogatz model, the time lost in preprocessing is not relevant enough that this process should be discarded.

**Table 3.4:** SEB calculation time for networks generated from the Newman-Watts-Strogatz model with and without preprocessing, time in seconds.  $n$ :number of nodes;  $k$ :joined with  $k$  nearest neighbours;  $p$ :probability of adding a new edge. Networks generated using the NetworkX package.

$n$	$k$	$p$	# edges	# bridges	Without preprocessing	With preprocessing
10	2	0.1	10	0	0.0064	0.0069
100	2	0.1	111	0	1.11	1.13
1000	2	0.1	1095	0	200	205
10	4	0.1	22	0	0.006	0.014
100	4	0.1	219	0	2.15	2.17
1000	4	0.1	2203	0	414	417
10	4	0.5	27	0	0.0070	0.0077
100	4	0.5	303	0	3.01	3.03
1000	4	0.5	2988	0	562	567

Another useful model to analyse is the Barabási–Albert model. The results for networks generated from such model can be seen in Table 3.5. As before the parameters were changed one at a time to better compare the results. It is possible to divide the results, and therefore the networks, into two categories:  $m = 1$  and  $m \neq 1$ .

When  $m = 1$  every edge is a bridge, so it is possible to draw conclusions regarding the first question: does preprocessing improve the calculation time when there are bridges present in the network? By looking at the table, the answer is clearly yes, improving the time significantly. This is even more noticeable when looking at the biggest network, when  $n = 1000$ , and notice that the calculation time with preprocessing is almost nonexistent when compared to without preprocessing.

When  $m \neq 1$  the number of bridges is always close or equal to 0. Once again, knowing that there are no bridges it is expected that the calculation time does increase, and in fact, that can be seen on the table. In this case the second question is appropriate: does preprocessing worsen the calculation time when the network has no bridges? Once again, it is possible to argue that the time lost with preprocessing is only a small fraction of the time spent with the calculation. Therefore it is possible to

conclude that with networks generated from the Barabási–Albert model, the preprocessing should also not be discarded. Based on these results it is reasonable to defend that preprocessing is always worth to be part of the calculation, because on the worst case, when there are close to 0 bridges, it does not affect significantly the calculation time, but in the best case it greatly reduces the computation time.

**Table 3.5:** SEB calculation time for networks generated from the Barabási–Albert model with and without preprocessing, time in seconds. n:number of nodes; m:number of nodes to attach from a new node. Networks generated using the NetworkX package.

n	m	# edges	# bridges	Without preprocessing	With preprocessing
10	1	9	9	0.007	0.002
100	1	99	99	1.012	0.0089
1000	1	999	999	185	0.08
10	2	16	0	0.004	0.006
100	2	196	0	1.24	1.29
1000	2	1996	0	463	488
10	4	24	1	0.006	0.006
100	4	384	0	2.49	2.51
1000	4	3984	0	729	760

Another potential improvement that was previously suggested was the use of *PySpark*, a Python interface for *Apache Spark*. The aim of using such interface is to parallelize the calculation, as edges are independent of each other. *Apache Spark* is a rather complex tool, with multiple modes that best-fit different use cases. The correct mode for this application is the cluster mode, however it is still complex to define the correct infrastructure for the machine that is being used. Therefore, in the first experiment the system configuration was not defined, letting the interface choose the best configuration. The results of this experiment can be seen in Table 3.6.

**Table 3.6:** Comparison between original serial calculation times to the use of *PySpark*, for each network. Time in seconds

Network	Serial	PySpark
PowerGrid	7936	26830
Facebook	1502	10109
Dolphins	0.60	21
C.elegans	307	3016
NetScience	78	1171
Web	3696	22306

By analyzing the results, it is obvious that *PySpark* is much slower than the original implementation, which can mean that this is not a good solution, or that it is necessary to try to better define the configuration, knowing the machine that is being used.

It is also important to mention that *Scipy*'s LAPACK module does already use parallelism in the

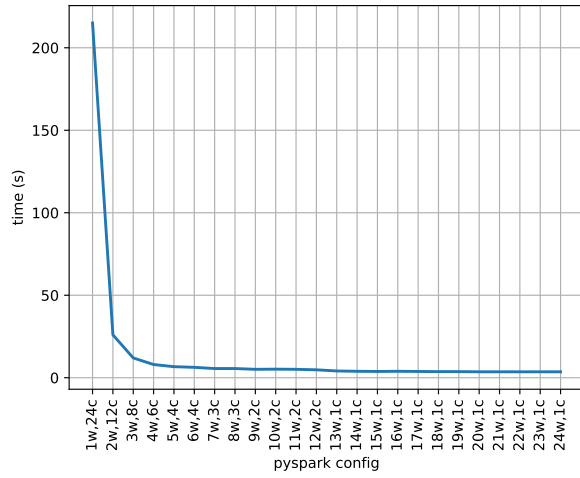
built-in function to perform the matrix decomposition. However, the implementation should be module agnostic. Because if to be replaced by another module or package that does not take advantage of parallelism, it is important to control the parallelism. This can be achieved by implementing it where it is more useful, such as the independent calculation for each edge.

To better take advantage of *PySpark*'s capabilities it is important to understand some of the inner workings of said interface. On cluster mode, the one suited for this application, Spark applications run as independent sets of processes, coordinated by the *SparkContext* object. To run on a cluster, the *SparkContext* connects to a cluster manager, in this case, the Standalone, a cluster manager included with Spark. After this connection is set up, Spark acquires executors on nodes in the cluster, that receive tasks from the *SparkContext* to run. Each executor in each worker node then has access to a certain number of cores. The number of worker nodes and the number of cores available to each one can both be defined.

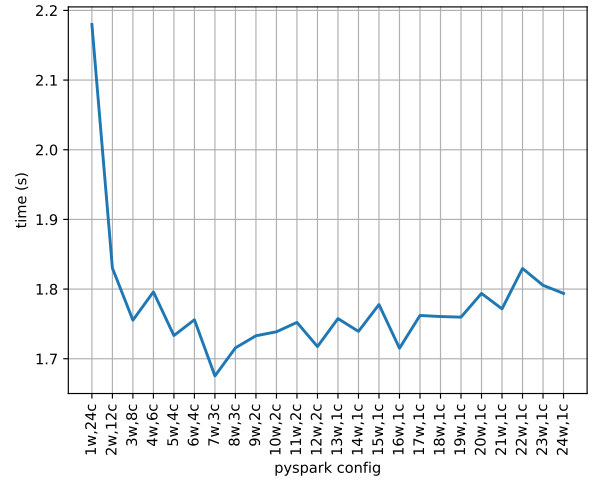
The machine used to perform these calculations has 24 physical cores, therefore the  $m = \#ofworkernodes \times \#ofcores$  should not exceed 24. Based on this number the next experiments consist in starting with 1 worker node and increase this number until 24, and adjust the number of cores each worker node has at its disposal such that  $m \leq 24$ . The results of such experiment can be seen in Figure 3.1.

By looking at Figure 3.1 it is possible to see a very abrupt decrease in time from 1 worker node to 2 worker nodes, and then in almost every network, a slower decrease until a somewhat stable value for the time is found. The case of the dolphins network, Figure 3.1b should be interpreted carefully, since every minor perturbation can disrupt the results by increasing just slightly the time. However, because the time is low in every different configuration those disruptions seem to be bigger than they really are. Resulting in some unexpected increases, but even with that the evolution of time is obvious and similar to all the other networks. The most obvious conclusion that can be made is that computation time decreases with increasing number of workers.

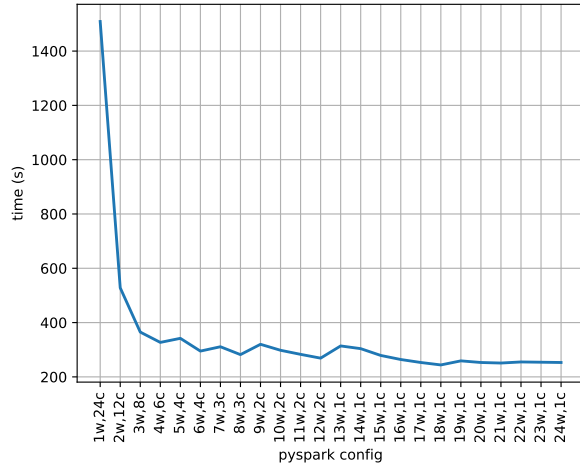
Another less obvious conclusion can be drawn by looking at Figure 3.1. The time, in almost every configuration, for all the networks except the dolphins network, is faster than the serial time. The dolphins network is the smallest, and is so small that the possible time won does not compensate the added complexity from parallelizing the calculation. However, when looking to the other networks, the best times show incredible progress in the quest to minimize the time spent in the calculations, showing that *PySpark* is a valid option, it just needs to be appropriately parameterized.



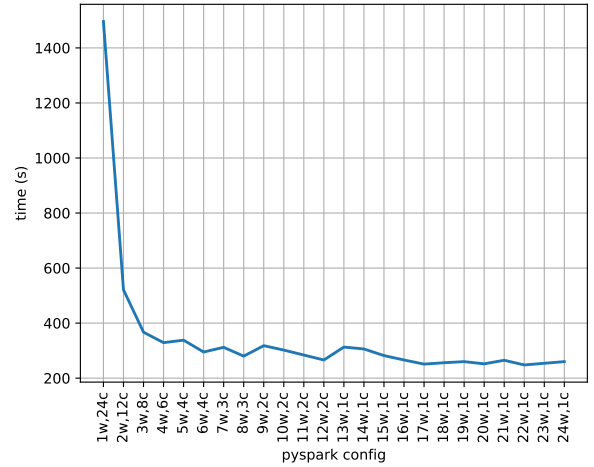
(a) C.elegans Network



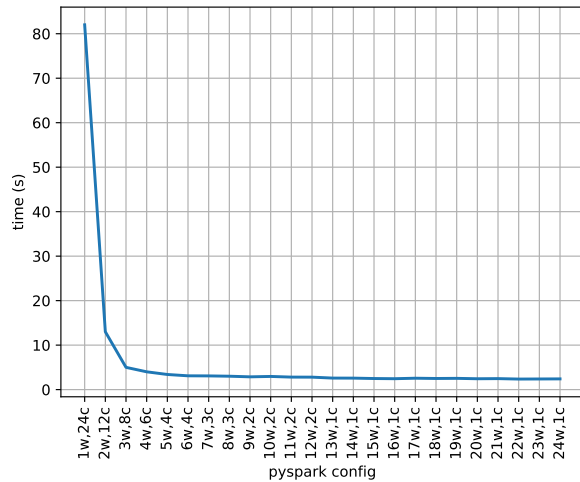
(b) Dolphins Network



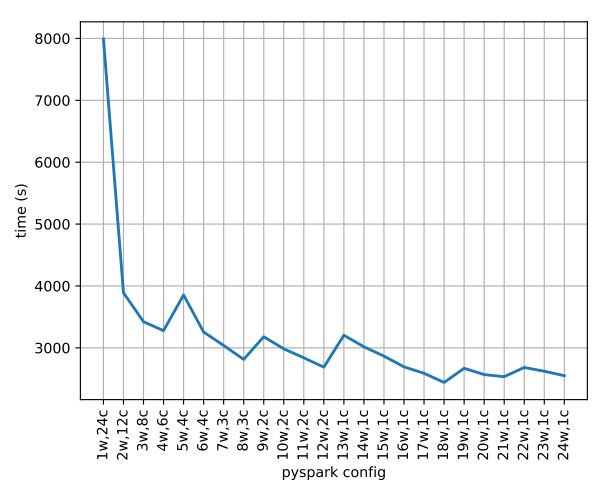
(c) Web Network



(d) Facebook Network



(e) NetScience Network



(f) Powergrid Network

**Figure 3.1:** Real world networks calculation times with multiple different *PySpark* configuration. w: worker nodes; c: cores.



# 4

## Spanning Edge Betweenness – Weighted Approximation

### Contents

---

4.1 Algorithm . . . . .	41
4.2 Results . . . . .	43

---



## 4.1 Algorithm

In the previous chapter we discussed some possibilities to try to reduce the computational effort, mostly related to time, spent when calculating the SEB values for a given graph. This was attempted by trying to divide the effort by multiple cores or by introducing preprocessing that could actively mitigate the calculation time.

Another way to reduce the computation time is by computing an approximation instead of the exact SEB values. This was done in [30] for unweighted networks. The authors took advantage of the relation between SEB and the effective resistance concept [61]. Effective resistance,  $R(e)$ , is calculated by looking to a graph as an electrical circuit and is equal to the probability that the edge is present in a random spanning tree of the graph [61, 62], therefore  $R(e) = SEB(e)$ . Effective resistance can be seen as pairwise distances between vectors [30]. This allows us to use the Johnson-Lindenstraus Lemma [63], as the pairwise distances are still preserved if we project the vectors into a lower-dimensional space. From the above observations we can extract Algorithm 4.1, that was first proposed by Spielman and Srivastava [64].

---

**Algorithm 4.1:** SEB approximations for unweighted networks.

---

```

1 begin
2    $Z \leftarrow [], L \leftarrow \text{laplacianMatrix}(G), B \leftarrow \text{IncidenceMatrix}(G)$ 
3    $Q \leftarrow \text{RandomProjectionMatrix}(k, m)$ 
4    $Y = QB$ 
5   for  $i=1 \dots k$  do
6     Solve  $Lz_i = Y[i, :]$ 
7      $Z = [Z; z_i^T]$ 
8   For every  $e=(u,v)$ , return  $R(e) = \|Z[u, :] - Z[v, :]\|_2^2$ 
```

---

By looking at Algorithm 4.1 it is possible to see that  $Q$  is the random projection matrix, and  $Y$  is where the projection applies. It allows to lower the dimensional space, because if instead of this we do  $Y = BB$  and  $k = m$ , there is no reduction in calculation cost. For the random projection matrix's entries it is possible to use either probability distribution mentioned in Theorem 2 from [65]. In this case the second option was chosen.

Another fundamental property is the value of  $k$ , and it can be calculated using:

$$k_0 = \frac{4 + 2\beta}{\epsilon^2/2 - \epsilon^3/3} \log(n) \quad (4.1)$$

where  $\beta$  controls the probability of success and  $\epsilon$  the accuracy in distance preservation, in other words, the error parameter. We can now define  $k \geq k_0$ , where  $k$  is an integer.

It is now possible to assure that with probability at least  $1 - n^{-\beta}$ , for all  $u, v \in E$

$$(1 - \epsilon)R(e) \leq ER(e) \leq (1 + \epsilon)R(e) \quad (4.2)$$

where  $ER(e)$  is the estimate of  $R(e)$ .

It is important to remember that is not always worth to use the projection to a lower-dimensional space, if the dimensional space is not lower after all, as it has to respect the formula present in equation 4.1. By looking at this equation it is possible to understand that  $k_0$  scales logarithmically with  $n$ , that is the number of nodes of the graph. However the multiplicative constant part of the equation should not always be disregarded. This part depends on both  $\epsilon$  and  $\beta$ , and depending on the values chosen for both variables the impact of this part will change accordingly. This can be called a threshold. Therefore, if the value of  $k$  does not represent a lower-dimensional space, it is better, from a computation cost point of view, to use  $Y = BB$ .

Moving on to line 6 in Algorithm 4.1, for a better understanding it is necessary to explain the notation used.  $Y[i, :]$  represents the  $i$ -th row of matrix  $Y$ .  $Z = [Z, z_i^T]$  represents the addition of column to matrix  $Z$ . As stated in [30] the aim is to compute  $QBL^P$  where  $L^P$  is the pseudoinverse of  $L$ , but if computed directly the computation cost is high, and does not offer any improvement over the exact measure calculation. Therefore in line 6 an approximation for this value is computed by approximately computing  $Lz_i = Y[i, :]$ . The solver used for this operation is a Symmetric Diagonally Dominant matrix (SDD) solver [66]. The result of this calculation is then stored in a placeholder matrix,  $Z$ . After this procedure has been repeated for every line of  $Y$ , it is necessary to compute the  $L_2^2$  distance between the vectors corresponding to nodes  $u$  and  $v$ , for every edge on the graph  $e = (u, v)$ .

We should keep in mind that it is possible to optimize this algorithm space-wise by instead of generating an entire matrix  $Q$ , in each iteration of the cycle a  $1 \times m$  vector,  $q$ , is generated accordingly to the distribution previously mentioned. In the same manner it is also possible to store the results in a placeholder  $1 \times m$  vector,  $z$ , calculating the SEB value for each edge iteratively.

A SSD solver is used to solve equations of the form  $Lx = b$  where  $L$  is a laplacian matrix, so it is symmetric, its off diagonals are non-positive, and all rows sum to 0, and  $b$  is in the span of the matrix.

The above method allows the calculation of the SEB values for an unweighted network. We can use this method to calculate the SEB values for weighted networks, with little adaptations. By looking at Algorithm 3.1 the necessary change is in line 13, where instead of calculating the exact measure, the method above is used, for the approximated measure. As in Algorithm 3.1 the SEB values are also calculated for each connected component and afterwards, each node inside each connected component is contracted into a single node.

By looking into the explanation above it is possible to conclude that this algorithm scales logarithmically with the number of nodes and linearly with the number of edges. Because of the way  $k$  is calculated it only makes sense to use the approximated version of the calculation when the number of edges,  $m$ ,

is high. Therefore if  $m$  is not high it is better to use  $Y = BB$  and  $k = m$ . In that case the complexity of the approximated measure is  $O(m^2 \log n \log \frac{1}{\epsilon})$ . The naive method has a complexity of  $O(n^3)$ , or even  $O(n^{2.5})$  if the theoretic limit is considered. So for the approximated method to compensate  $m^2$  has to be inferior to  $n^3$  (or  $n^{2.5}$ ) and that only happens if  $m = O(n)$ , i.e., if the graph is sparse.

## 4.2 Results

As stated before, the primary reason for the introduction of an approximated measure is to reduce the computational time. Theoretically the threshold for the approximation measure to be better than the exact measure was stated before to be considerably high, as it increases with the accuracy of the results and becomes more irrelevant the more nodes a network has. It is however important to have empirical results. This results can be seen in Table 4.1, where there is a comparison between the time spent for the calculation of the exact measure and for the approximated measure. The networks used are all real world unweighted networks and some of their properties can be consulted in Table 3.1.

**Table 4.1:** Comparison of times between approximated and exact SEB values, time in seconds.  $\epsilon = 0.01$  and  $\beta = 1$ .

Network	exact	approximated
PowerGrid	7936	7069
Facebook	1502	1303
Dolphins	0.60	51
C.elegans	307	426
NetScience	78	259
Web	3696	33302

Before looking into Table 4.1 and the results it shows it is crucial to mention that for reasons of integration into *Python*'s network libraries the entire implementation of the exact measure is done in the programming language *Python*. On the other hand, for the approximate implementation, the SDD solver<sup>1</sup> was only available in the *Julia* programming language, therefore the calculation of the SEB values for each connected component is performed using *Julia*, that is known to be faster than *Python*, however the rest of the algorithm is still done in *Python*. This difference obviously has an impact in the calculation time, but in spite of that, conclusions can still be made.

By analyzing the results in Table 4.1 we can see that there are very different results from network to network, i.e., there are networks where the time got significantly worse, there are networks where the time got a little worse and then there are networks where the time improved. This calls for a deeper analysis of the results, where it is taken into consideration the properties of the network. These properties can be seen in Table 3.1 where it is possible to observe for each network the number of nodes and

<sup>1</sup><https://github.com/danspielman/Laplacians.jl/blob/master/docs/src/usingSolvers.md>

edges, the clustering coefficient, the assortativity and finally the average degree.

For the *Dolphins* network, the increase in time is expected, because of the constant calculation time, that is more noticeable in smaller networks, as stated before and as observed here, with the other characteristics of the network not having much impact on the calculation time. The other network which also shows a far worse result is the *Web* network. This network has a high number of nodes and edges, but also a high clustering coefficient value. The threshold where it is better to approximate, for the values used for  $\epsilon$  and  $\beta$ , is quite high, so it is expected that the time spent would be worse than for the exact calculation, and together with the high value for the clustering coefficient can justify the increase in running time. Both networks show a nearly 10 times worse calculation time for the approximated measure than for the exact measure, but for different reasons.

Moving on to the networks where the time got slightly worse, there are the *C.elegans* and *NetScience* networks. Both networks are not big enough that it should compensate to use the approximated measure and by looking at the running time it is possible to see that, indeed, it does not compensate. However, the time does not get worse by the same factor as the case aforementioned. In this case there is a factor of 1.5-3 in the time difference. It is expected that this time factor is inversely proportional to the size of the networks, i.e., as the size gets bigger the factor tends to become smaller, as it gets closer to the threshold where the approximated measure is more worth to use. It is also worth to point out that the clustering coefficient for both networks is pretty high, which is another reason for the time to be so worst when compared to the time from the exact SEB values calculation.

Finally, but not least, there were networks in which the time improved. This happened for the *Power-Grid* and *Facebook* networks. These networks are both in the range of thousands of nodes and edges, so it is possible to affirm that they have a considerable size. It is necessary to interpret this results carefully. As mentioned before the calculation part of the approximated measure is done in the *Julia* programming language, that is faster than *Python*, and that definitely contributes to have a lower calculation time than if everything was done in *Python*, however if we are not over the threshold we are definitely close to it, and it is lower than expected. This can be justified by the low clustering coefficient that both networks have.

The networks used for these results are all unweighted, however the conclusion about the importance of the clustering coefficient can be extended to weighted networks but for the giant connected component formed with edges all with the same weight. This and the relative size of the giant component to the rest of the network are two of the most important characteristics that contribute to lower the threshold where it is worth to use the approximated measure. This can only be seen for networks with somewhat homogeneous weights, so it is possible that a giant component exists, because otherwise if there is too much heterogeneity in the weight values a giant component will never be formed. In this later case the time would then increase linearly with the number of edges, but with the addition of the constant part,

and this would never go beyond the threshold, making it always worth to use the exact measure. This topic needs further investigation, and unfortunately there was not enough time for it now, but it is left as a suggestion for future work.



# 5

## Edge Percolation

### Contents

---

5.1	Number of Connected Components . . . . .	50
5.2	Size of the Giant Connected Component . . . . .	52
5.3	Diameter of the Giant Connected Component . . . . .	54
5.4	Median of Connected components' size . . . . .	57
5.5	Average path length inside the giant connected component . . . . .	59
5.6	Number of connected components and the giant's connected component size . . . .	61
5.7	Conclusions for each network . . . . .	63

---



For context is important to clarify two aspects related to percolation. In many backgrounds percolation represents the addition of structures to a network, and reverse percolation the removal of structures. In this work, specially in this chapter the word percolation is used to represent the removal of structures from a network. The second aspect is that percolation is usually related to nodes, but in this work and specially in this chapter, edges are the removed structures, as SEB provides a value of centrality for the edges.

In this section edge percolation was performed on real world networks. It is important to recall the networks' characteristics, that can be seen in Table 3.1. The aim of this chapter is to compare SEB to Edge Betweenness Centrality(EBC) when percolation is performed on a network based on the value that both these centrality measures attribute to each edge.

In each plot there are 4 lines, where 2 represent SEB and the other 2 represent Edge Betweenness Centrality. Out of those two lines for each measure, one represents the measure with recalculation after removing an edge and the other represents the measure without recalculation between iterations. This is interesting as it allows to draw conclusions if the recalculation can have an impact and if so which metrics does it impact the most. Speaking of metrics, there are 5 metrics that were recorded. First is the number of connected components; Second is the size of the giant connected component; Third is the diameter of the giant connected component; Fourth is the median of the connected components' size; Fifth is the giant's connected component average path length. All of the plots mentioned before have each metric represented in the  $yy$  axis and the percentage of edges removed on the  $xx$  axis. Last but not least there is another plot that relates the number of connected components with the size of the giant connected component. Each figure on this chapter contains the aforementioned plots grouped by metric, therefore each subfigure contains the results for said metric for a specific network.

It is important to take into consideration each network's characteristics for a better analysis of the results, which can be done by consulting Table 3.1. The results for the *NetScience* network are represented in every (a) subfigure. This network is not particularly big, however its number of edges is almost the triple of the number of nodes, which then leads to an high clustering coefficient. In every subfigure (b) it is possible to see the results for the *C.elegans* network, which has a relevant size, with only a few hundred nodes but a couple thousand edges, also with an high clustering coefficient, and the highest average degree of all the networks, which is expected when there are 5 times more edges than nodes. Now moving on to subfigures (c), they show the results for the *Facebook* network, which can be considered to have a relevant size, with almost three thousand nodes and edges, with a very low clustering coefficient, and an average degree of 2, which is expected when the number of nodes is almost equal to the number of edges. Subfigures (d) show the results for the *Dolphins* network. This network is very small, with only around half a hundred nodes and one hundred and a half edges, also with a quite low clustering coefficient, and an average degree of 5. In subfigures (e) are the results for the *Web* network.

This network is the second largest used in this study, with over three thousand nodes and six and a half thousand edges, also with a quite high clustering coefficient, and an average degree of 4.

## 5.1 Number of Connected Components

The first metric to be analyzed is the evolution of the number of connected components. This evolution, for every network, can be seen in Figure 5.1.

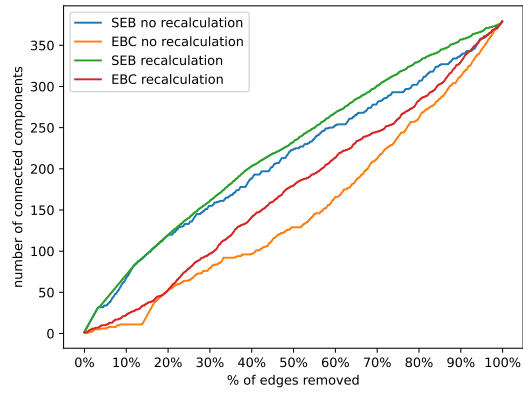
By looking at Figure 5.1a it is possible to observe a clear difference between SEB and EBC, and even between EBC with recalculation and EBC without recalculation. In the early stages of percolation, both SEB variations have similar results in the number of connected components that each produces, but as more nodes are removed, specially after around the 60% value, there starts to exist a slight difference between these two metrics. Regarding both EBC variations, it is possible to see that at the start they are quite similar, and then from around the 25% spot to the 75% spot a difference starts to emerge and then, when reaching the end of this interval, they come back to be very similar again. It is also observable that at the end, after about 90% of nodes have been removed, that SEB without recalculation and EBC with recalculation produce the same amount of connected components.

Moving on to Figure 5.1b it is possible to see that there is a big gap between different centrality measures. Both SEB variations are quite similar for the whole duration of the process and both have a close to linear growth in the number of connected components they produce. In fact, also both EBC variations are quite similar during the whole percolation process, but they generate a very few number of connected components until around the 55% mark for EBC with recalculation and around 70% for EBC without recalculation, but after this point both show a big growth until the end of the percolation.

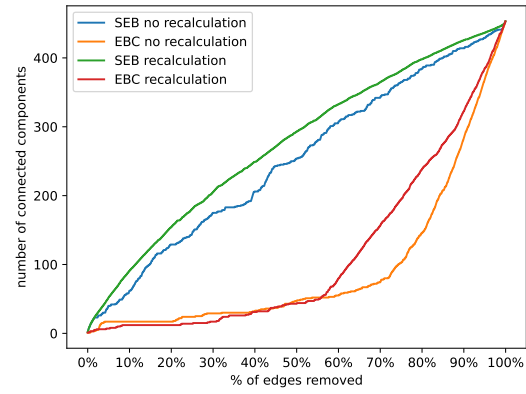
By looking at Figure 5.1c, it is possible to see that there is no real difference between any of the measures and neither the variations, as they all show a linear growth.

In Figure 5.1d is perceivable that SEB variations start quite similar and gradually diverge more and more to the end of the percolation process. EBC variations are, however, similar for only a few iterations at the very start, then proceed to diverge, to then switch places on which generates the lower number of connected components until they converge, very close to the end.

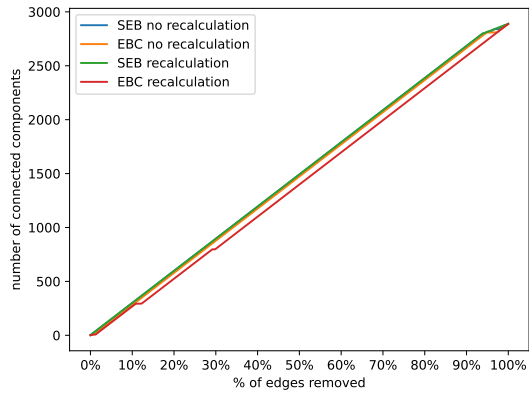
By looking at Figure 5.1e it is possible to see that there is a big difference between both SEB variations in this metric, with them starting quite similarly and then quickly being very far apart to then converge again very close to the end. Regarding EBC variants, they cannot be considered similar but the gap is never as big as it is between SEB variants, however they are only similar very close to the end, after the 90% mark. SEB with recalculation achieves higher values all the way through the percolation process. SEB without recalculation almost always achieves higher values than both EBC variations.



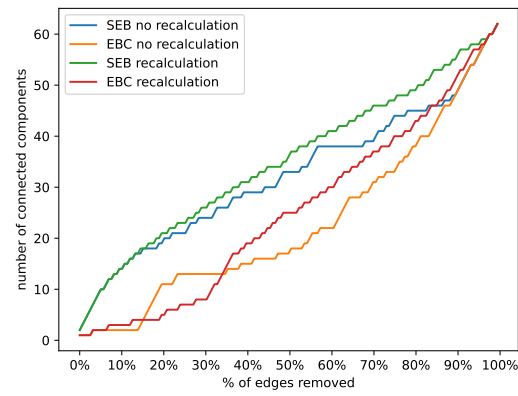
(a) *NetScience* network.



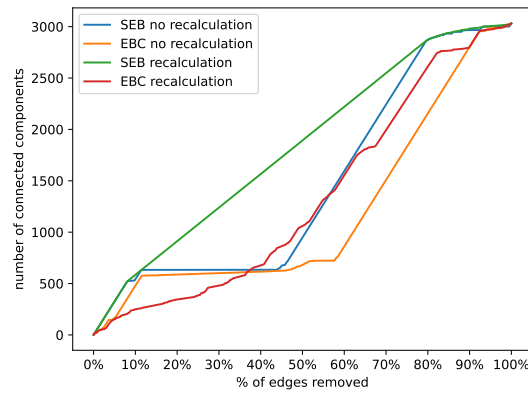
(b) *C.elegans* network.



(c) *Facebook* network.



(d) *Dolphins* network.



(e) *Web* network.

**Figure 5.1:** Evolution of the number of connected components with the removal of edges, for every real world network studied.

It is curious to see that the most similar plots are the ones from *NetScience* and *Dolphins* networks, even if these networks have very little in common. In fact, from the characteristics present in Table 3.1,

the only one that are close is the assortativity and the number of edges being approximately two and a half times the number of nodes, and in this case, that seems to be enough to produce similar results. In every network except *Facebook*, there are noticeable differences between the centrality measures used, but not so much between variations inside each centrality measure, which can indicate that there is no great benefit from using the most computationally expensive variation(with recalculation).

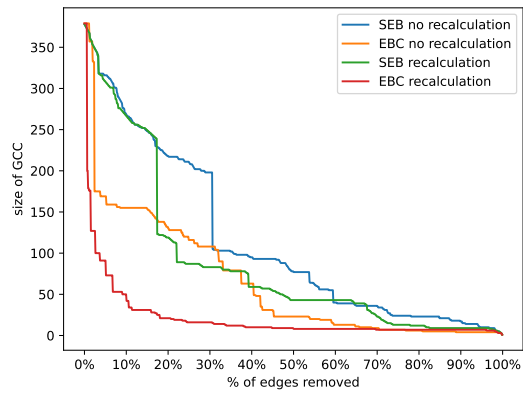
## 5.2 Size of the Giant Connected Component

The second metric to be analyzed is the evolution of the size of the giant connected component. This evolution, for every network, can be seen in Figure 5.2.

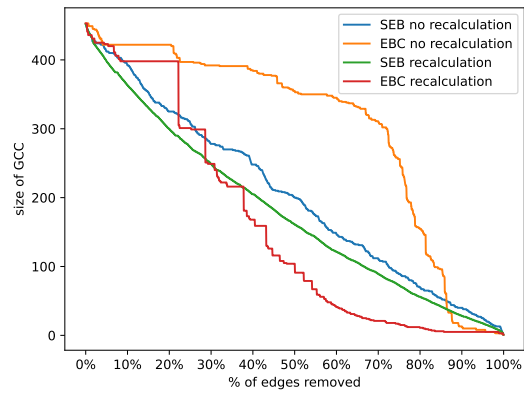
In Figure 5.2a there is also relevant differences between SEB and EBC and at some points also between with and without recalculation inside the same centrality measure. Looking into both SEB variations it is possible to perceive that they are pretty similar until the 20% point at which a big gap emerges, where SEB with recalculation shows a lower size of the giant connected component, that then gradually gets smaller until they back to being similar at about the 60% of nodes removed mark. Now about EBC, it also starts quite similar until about the 5% mark where they split. The EBC with recalculation variation continues the trend coming into this mark and keeps dropping the size of the giant connected component until it stabilizes at the 10% mark with a size of 25 nodes, and EBC without recalculation only catches up to this size value at around the 45% mark, from where both variations are very similar until the end. In general EBC achieves a lower size for the giant connected component than SEB throughout the whole process of percolation, however there is a small gap, where SEB with recalculation shows a lower size for the giant connected component than EBC without recalculation, and this happens from around the 15% mark up until the 40% mark.

At a first glance to Figure 5.2b it is possible to see that each measure behaves differently than each other, and also that there is a big difference between variations of the EBC measure. It is possible to see that both variations of the SEB measure start and stay quite similar all the way to the end. On the other hand there is a very big difference between each variation of the EBC measure, that starts around the 20% point. They show opposite behaviour from each other, as EBC with recalculation drops the size of giant connected component quite a lot after this point, EBC without recalculation produces a close to constant giant's connected component size, only dropping the same as the other variant a lot later.

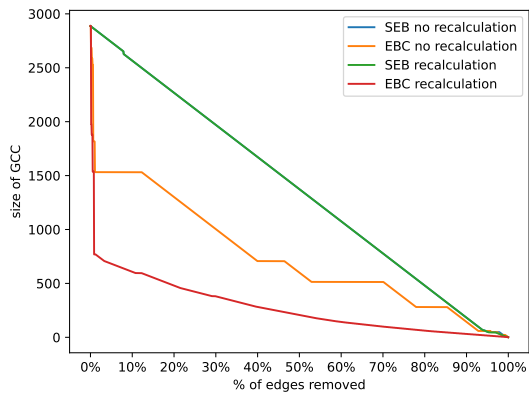
Figure 5.2c shows that both SEB variations still are an almost perfect match to each other, but the same cannot be said to EBC variations, as they are different from one another almost the entire duration of the process. SEB with recalculation starts by dropping the giant's connected component size from almost 3000 to around 750 nodes, whereas SEB without recalculation only drops the size by close to half of that. EBC with recalculation then slowly decreases the value until the end of the process.



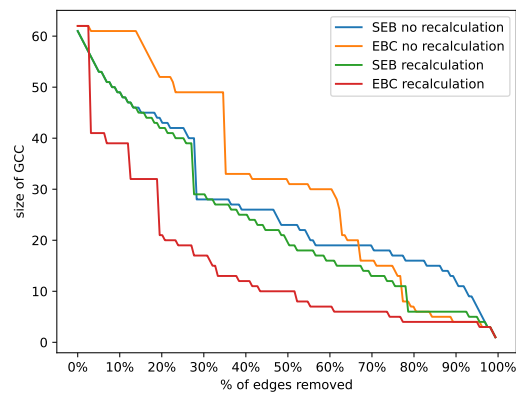
(a) *NetScience* network.



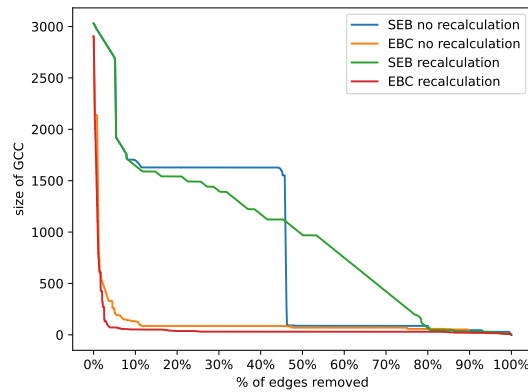
(b) *C.elegans* network.



(c) *Facebook* network.



(d) *Dolphins* network.



(e) *Web* network.

**Figure 5.2:** Evolution of the size of the giant connected component with the removal of edges, for every real world network studied.

Now moving on to Figure 5.2d it is possible to see the results for the evolution of the size of the giant connected component. Once again, the similarity between both SEB variants is clearly higher than

between EBC's variants. In fact, both SEB variations start quite similar and stay like that for half of the process, and then start to differ more and more by each iteration, quite similar to the previous plot. On the other hand, EBC variations differ quite a lot throughout the whole process, only converging near the end, at the 90% mark.

Looking into Figure 5.2e it is possible to see big differences between both centrality measures. Both EBC variations are very quick to drop the size of the giant connected component, where SEB variations are much slower, taking at least half of the process to reach the same values, where SEB without recalculation has a huge drop at about 45% mark and SEB with recalculation has a more linear decrease in the value it produces. It is also important to mention that EBC with recalculation is the variation that produces the lower values, after the first iterations all the way to the end.

The results can be grouped into two groups: the one where both SEB variations produce the highest values and the one where EBC without recalculation provides the highest values, obviously not for the entirety of the process, but for the major part. Even though this similarities exist, they are only for relative values, because the lines display different behaviours between different networks even if the relative order is the same. There are bigger differences between variations for the EBC centrality measure than for the SEB centrality measure.

### 5.3 Diameter of the Giant Connected Component

The third metric to be analyzed is the evolution of the diameter of the giant connected component. This evolution, for every network, can be seen in Figure 5.3.

In Figure 5.3a it is possible to see that there is a big difference between both variations of EBC, that starts right at the beginning and lasts almost all the way to the end. This plot is a good example on the impact the recalculation or not of a measure after each iteration can have to a metric. In fact, the impact is so substantial that EBC without recalculation results in the lowest giant's connected component diameter out of the 4 variants and EBC with recalculation shows the biggest giant's connected component diameter for close to 50% of the percolation process. For the SEB variations, they start quite similar up until around the 20% mark, but after, there is always some difference between them, with SEB with recalculation being the variation that shows lowest values for the diameter of the giant connected component, inside the SEB centrality measure.

By looking at Figure 5.3b is perceivable that both variations of the SEB measure show similar results, and both produce almost always lower diameter values than the EBC variations. Alternatively to that, it is observable that EBC variants are not at all similar. EBC with recalculation shows a behaviour that is more similar to both SEB variants, starting off with an average value and after half the edges have been removed, gradually decreases until the end. On the other hand EBC without recalculation starts with an

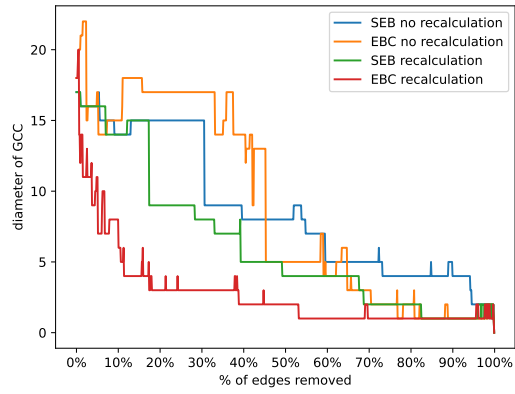
average value, which then increases after the 50% mark and then suddenly drops at around the 85% point. There are a lot of spikes on this plot, that at a first glance may seem kinda odd, but they can be explained by the change of the giant connected component, where after an iteration there is a change on the connected component that has the biggest size, and therefore there is a sudden change in the diameter value.

In Figure 5.3c there is no real difference between both SEB variations. On the other hand, EBC variations are quite different throughout almost the entire percolation process. EBC with recalculation quickly drops the diameter to 2, then stays constant for 99% of the process and then right at the end drops to 0. EBC without recalculation goes through some ups and downs but never goes lower than the other EBC variant.

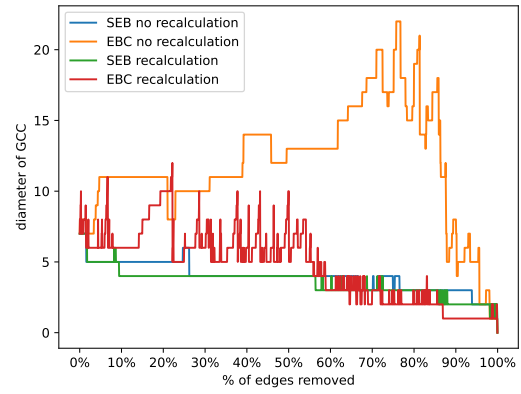
Looking into Figure 5.3d, the SEB variations are quite similar for the majority of the process and EBC variations not so much. As can be seen, EBC with recalculation produces the lower diameter values, where on the other hand EBC without recalculation produces the higher diameter values out of the 4 variants. Both SEB variants and SEB with recalculation produce average to low diameter values, where EBC without recalculation produces average to high diameter values, for the greater part of the process.

In Figure 5.3e, once again, both SEB variations are quite similar, producing very low values when compared to EBC without recalculation, that produces the highest values by far. EBC with recalculation produce values much closer to SEB, but shows a different behaviour, where it starts with a lot of spikes, and then drops to a very low value and has some more spikes at the end. All the other variations show little to no spikes.

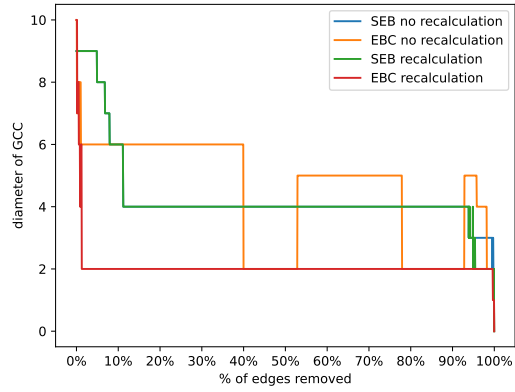
From all the plot in Figure 5.3 it is possible to conclude that this metric produces relative behaviours similar for all the networks. EBC without recalculation is always the measure who achieves the highest values. Both SEB variations are quite similar throughout the entire process, and finally, EBC with recalculation always produces the lowest values for at least a considerable part of the process. It is also arguable that, for at least EBC, recalculation has a big impact in the results obtained, however for percolation purposes, the higher the diameter is the better, and if before there was a doubt regarding the benefit/cost relationship of performing recalculation, for this metric there is absolutely no doubt, that for every network used here, it is not worth to spend more computationally because the results achieved are worst.



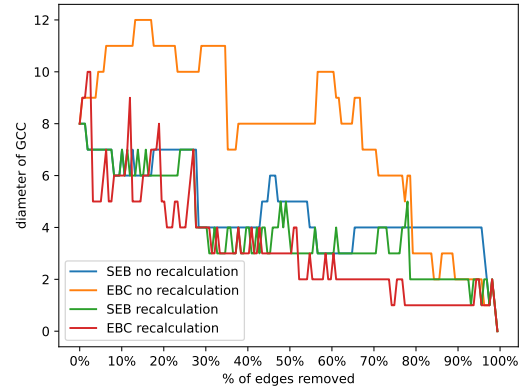
(a) *NetScience* network.



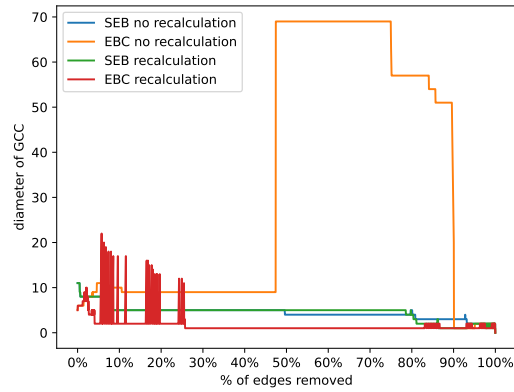
(b) *C.elegans* network.



(c) *Facebook* network.



(d) *Dolphins* network.



(e) *Web* network.

**Figure 5.3:** Evolution of the giant's connected component diameter with the removal of edges, for every real world network studied.

## 5.4 Median of Connected components' size

The fourth metric to be analyzed is the evolution of the median of the connected components' size. This evolution, for every network, can be seen in Figure 5.4.

By examining Figure 5.4a it is possible to see that there are no big differences between all variations. However there are some worth pointing out. There is a slight difference between SEB and EBC results. SEB starts at half the EBC value, then quickly drops to 1 and stays that way the entire process. EBC starts at a higher value, as was just stated, then drops for a few iterations to the initial SEB value. So far both iterations are very similar, however some differences start to emerge. They both show a curvature, when dropping to the value 1, however EBC without recalculation reaches this value at the 15% mark, but EBC with recalculation only reaches this value at the 35% mark.

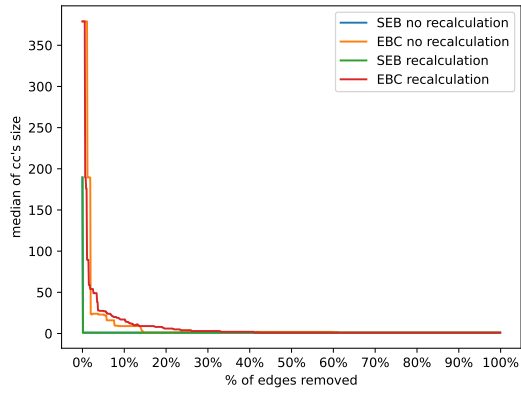
In Figure 5.4b it is possible to see that there are no big differences between any of the 4 variations as they all quickly go to 0 or very close to it in case of the EBC with recalculation variant. This shows that for this network all variations very fast create a majority of single node connected components, contrary to what was observed on the previous network.

By looking at Figure 5.4c it is possible to see that once again there is no real difference between any of the 4 variations displayed, if not the one that can be seen in the very beginning of the process, where it is possible to perceive that EBC variations take a little more iterations to get to the 1 value, where EBC with recalculation is the slowest, then comes EBC without recalculation, and SEB variations are equally quick to get to the 1 value.

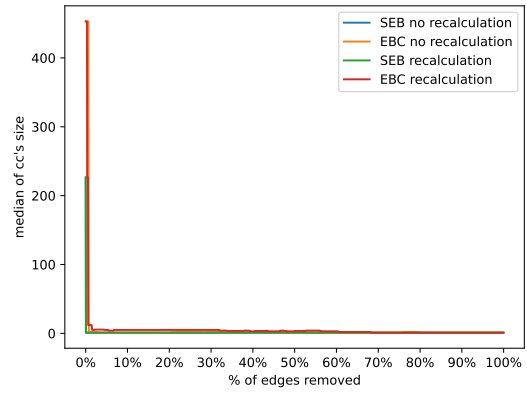
In Figure 5.4d at first sight this plot might seem drastically different from the ones from the previous networks. However, after a closer look it is possible to perceive that both SEB variations immediately go to 1, and EBC variations take a little while longer to get there. Because this network is so small compared to the previous ones it may look that it takes many iterations for EBC variations to get to 1, but this may as well be the same value of iterations that took on the previous networks, but because they were big, this number of iterations represent a much less percentage of the whole process.

The results in Figure 5.4e are pretty similar to the ones from Figure 5.4b and 5.4c, where both centrality measures show some differences in the early stages, but then they are equal all the way to the end. There is also no difference between variations inside each centrality measure. The only difference between centrality measures is that EBC only converges to 1 at about the 5% mark.

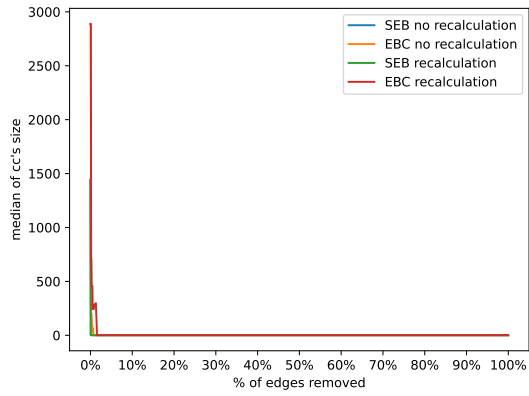
The conclusions that can be drawn after analyzing the plots from Figure 5.4 are that there is no big difference between networks for this metric. Also there is no big differences between with or without recalculation, where in the previous metrics they were clear, and when both variations produce the same, or very close, values it is never worth to use recalculation as it increases by a lot the computational cost.



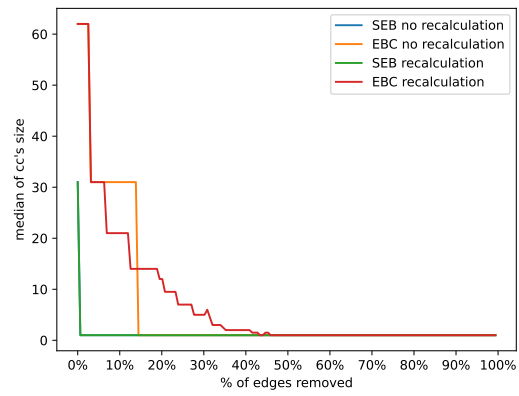
(a) *NetScience* network.



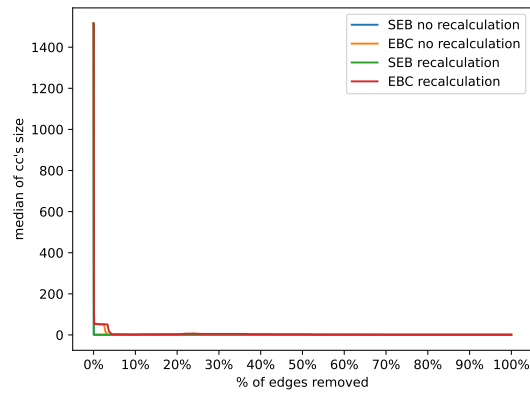
(b) *C.elegans* network.



(c) *Facebook* network.



(d) *Dolphins* network.



(e) *Web* network.

**Figure 5.4:** Evolution of the median of the connected components' size with the removal of edges, for the real world networks studied.

## 5.5 Average path length inside the giant connected component

The fifth and last metric to be analyzed is the evolution of the average path length inside the giant connected component. This evolution, for every network, can be seen in Figure 5.5.

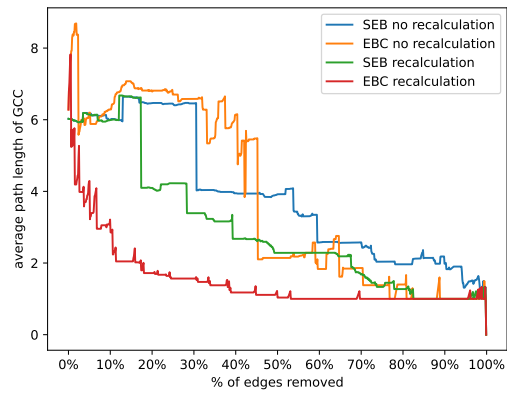
Figure 5.5a shows very similar results to Figure 5.3a. A big difference between both EBC variations and a smaller between both SEB variations also exists here. Likewise in Figure 5.3a, also here EBC with recalculation shows the lower average path length value, with EBC without recalculation showing the highest value for almost half of the percolation process. In the same manner SEB without recalculation also shows higher values than SEB without recalculation.

In Figure 5.5b as in the previous network there are similarities between Figure 5.3b and Figure 5.5b, but in the latter the changes are not as sudden. Both SEB variants are quite similar throughout the entire percolation process, starting at an average length and slowly going down. On the other hand, there is a big difference between both EBC variations, where the one with recalculation trends in the same direction as the SEB variations, and the one without recalculation also starts at an average length value, but it instead of going down, starts to go up hitting the highest value at around the 85% mark and then going down, catching up with the rest of the variants.

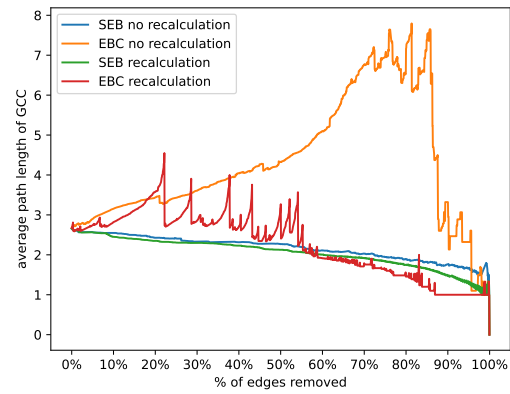
Figure 5.5c can once again be compared to Figure 5.3c, where both SEB variations are a perfect match for almost the entirety of the percolation process, and alternatively to that, the EBC variations barely match throughout the process, with EBC without recalculation producing the lowest values of all the variants and EBC with recalculation has some ups and downs until almost the end where it catches back up with every other variation.

Looking into Figure 5.5d it is possible to see that both SEB variations start quite similar and as the percolation process occurs they start to grow apart. Both variations produce intermediate values. For the EBC variations they are, again, very different from each other, where EBC with recalculation achieves the lowest values and EBC without recalculation produces the highest values. After the 80% mark, SEB without recalculation does take this later spot and produces the highest values, until the end.

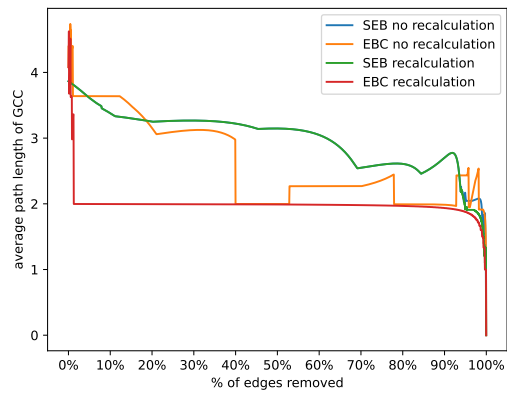
Figure 5.5e can also be compared to Figure 5.3e because both show very similar behaviour. Once again both SEB variations stay similar throughout the whole process and produce results in the middle of EBC variations. On the other hand, EBC variations are not similar at all, with EBC with recalculation producing the lowest values and EBC without recalculation producing the highest values. In fact, there is a time during the process where EBC without recalculation produces values that are at least four times bigger than the maximum values produced by all the other measures. This happens from the 50% mark up until the 90% mark.



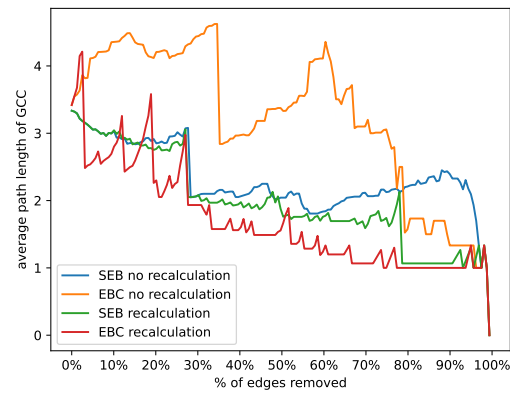
(a) *NetScience* network.



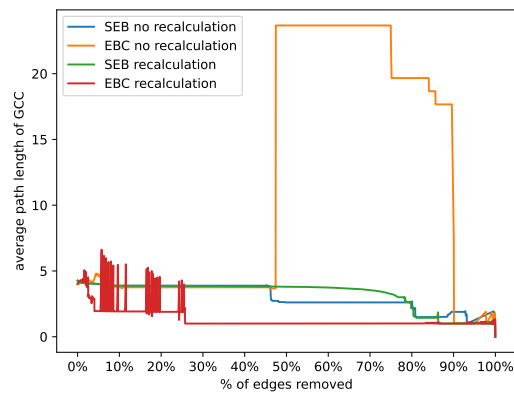
(b) *C.elegans* network.



(c) *Facebook* network.



(d) *Dolphins* network.



(e) *Web* network.

**Figure 5.5:** Evolution of the average path length inside the giant connected component with the removal of edges, for the real world networks studied.

To conclude, this metric shows very similar results to the diameter, in Figure 5.3. There is no much impact using recalculation for the SEB centrality measure, as the results produced are similarly enough

to not be worth the extra cost of applying recalculation between every iteration. Contrary to that, EBC variations show a big difference between themselves. However, for this metric, from a percolation standpoint a better result is the one that produces higher values, and EBC without recalculation is the one that achieves that from the two variations, so it is possible to say that, without a doubt, EBC with recalculation is not worth its cost. The results are similar, when looking at the relative values, to all the networks, except for *Facebook*.

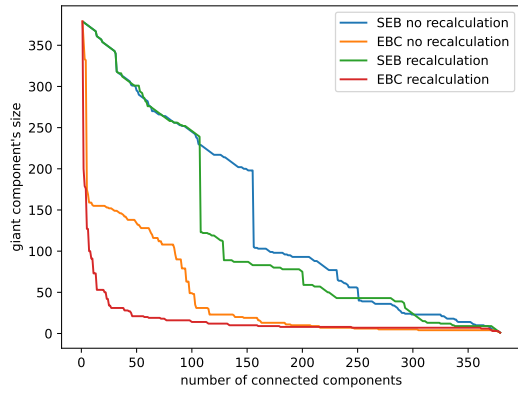
## 5.6 Number of connected components and the giant's connected component size

In Figure 5.6a there is a substantial difference between SEB variations and EBC variations. For the SEB variations they start identical, and stay like so until there are 100 connected components, after which SEB recalculation suddenly decreases by a lot the giant's connected component size while SEB without recalculation only does this about 50 connected components after. They then stay quite similar all the way to the end. For the EBC variations stay similar for the initial and final part of the percolation process, but the key difference to the SEB variations is that they require the formation of much less connected components to effectively break the giant connected component.

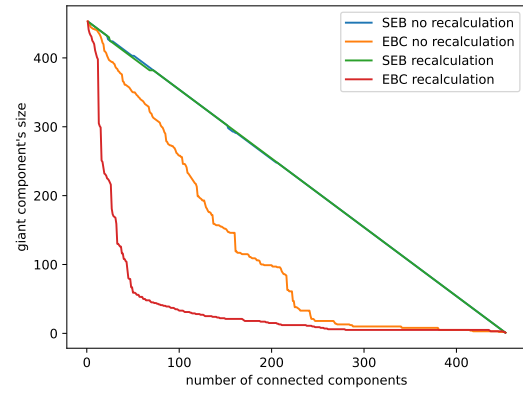
In Figure 5.6b is possible to perceive that both SEB variations are almost perfectly linear, where EBC variations are not and are also very different from one another. EBC with recalculation drops the size of the giant connected component a lot with the formation of only a few connected components and after this big drop it goes very slowly down all the way to the end, where EBC without recalculation is slower to achieve the same giant's connected component size, and only gets there halfway through the percolation process.

As in Figure 5.6c the evolution of the number of connected components is linear, this figure is basically the same as Figure 5.2c. EBC variations produce less connected components while lowering the giant's connected component size, and inside the EBC measure, the EBC with recalculation variant is the one that achieves the lower value for the size of the giant connected component.

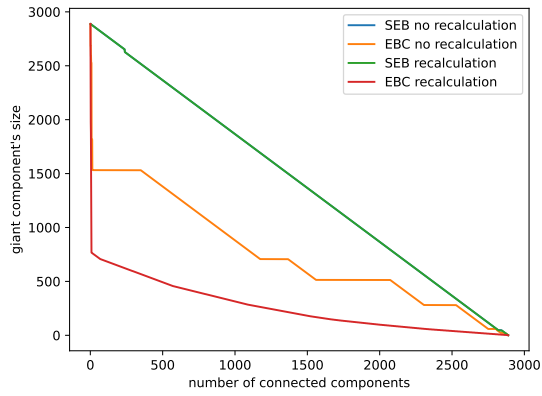
In Figure 5.6d both SEB variants are pretty similar from start to end of the process, but not only that, also EBC without recalculation starts very similarly to both SEB variations, until the moment that it achieves a lower giant's connected component size, while creating a lower number of connected components. As always, EBC with recalculation stands out by achieving the lowest size for the giant connected component while also creating the fewest connected components.



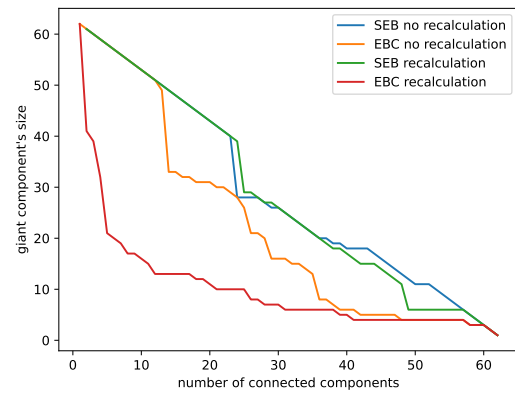
(a) *NetScience* network.



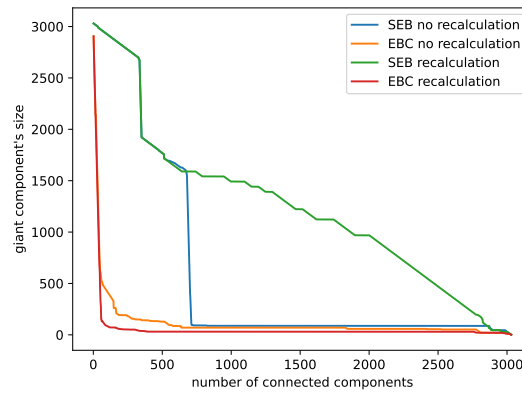
(b) *C.elegans* network.



(c) *Facebook* network.



(d) *Dolphins* network.



(e) *Web* network.

**Figure 5.6:** Relation between the evolution of the number of connected components with the evolution of the giant's connected component size, for all the real world networks studied.

Figure 5.6e shows interesting results. Starting by the EBC variations, they are quite similar for the whole process. After only a few connected components were formed, the size of the giant connected

component is already very low, with there being a small difference between variations in the early stages of the process, but they quickly disappear and then they stay close to equal for the rest of the process. SEB variations start equal, and need more connected components to decrease the size of the giant connected component, however around the 750 connected components point, SEB without recalculation drops the size of the giant connected component by a lot, equalling the values from EBC variations. On the other hand, SEB with recalculation decreases the size of the giant connected component in a slower way, while creating more connected components than all the other variations

To conclude, it is possible to see a pattern in every plot from Figure 5.6. SEB variations always produce an higher value for the number of connected components for the same size of the giant connected component value. SEB variations are also quite similar throughout the whole process for all the networks, except for *Web*. EBC variations always produce lower values for the size of the giant connected component for the same number of connected components. Even though variations are also quite similar, there is a bigger difference between EBC variations than SEB variations, except again for *Web*. EBC with recalculation always achieves the lowest values, even for *Web*.

## 5.7 Conclusions for each network

From all the plots in subfigures (a) and the analysis made above, conclusions can be draw, regarding the differences between centrality measures and even between variations inside the same measure, for the *NetScience* network. It is possible to state that there are differences between both centrality measures, where SEB originates a bigger number of connected components but it is not that effective at breaking the giant connected component, and on the other hand, EBC does not generate as much connected components as quickly as SEB but it does break the giant connected component much faster, specially on the recalculation variation. It is also possible to see the impact of recalculation or not of a measure, specially in the metrics within the giant connected component, however it is important to have in mind the cost/benefit relationship to decide if it is worth or not to recalculate between each iteration, which can be worth for some metrics but not for some others.

To conclude, by looking at all the plots from subfigures (b), it is possible once again to point out differences between both measures, because also in this *C.elegans* network SEB variations are better at creating more connected components, however EBC variations form less connected components while more effectively lowering the giant's connected component size(Figure 5.6b) but only the most computationally expensive EBC variation achieves a lower giant's connected component size for the same number of edges removed as the SEB variations(Figure 5.2b), and only after the 25% mark. Also for this network EBC variations achieves higher values for metrics taken inside the giant connected component. As before, there is a bigger difference between variants in the EBC measure than in the

SEB measure.

The *Facebook* network can be fit into the scale-free network's type, as it has a few very high degree nodes and the majority are low degree nodes, which can be seen in the very high negative assortativity value. Based on the plots from subfigures (c), it is possible to conclude that there is basically no difference between EBC and SEB regarding the creation of connected components, however there is a big difference regarding the giant's connected component size, with EBC measure being better at breaking it down. For the metrics inside the giant connected component there are differences to the previous networks. For the diameter one variation of EBC produces higher values for the bigger part of the process and the other produces lower values for the entire process. The variation that produces the lower values is EBC with recalculation, that is also the more expensive to calculate, out of the two. For the average path length both SEB variations produce an higher value for most part of the process than both EBC variations.

When applying percolation to such a small network, like *Dolphins* is, weird and unexpected behaviours could occur, since every little aspect has more impact than when in a bigger network. With that being said and comparing the plots from subfigures (d) and comparing them to the previous networks, it is possible to say that the results are quite similar to the bigger networks, namely the *NetScience* network. Once again there are key differences between SEB and EBC, where SEB is better at disintegrating the network into multiple connected components and EBC is better at breaking the giant connected component apart. Also as before, there is a big impact, specially in the metrics inside the giant connected component, between recalculation or not of a measure between every iteration. There is no big difference between both SEB variants in any of the plots, however the same cannot be said for the EBC variants, where there is a big difference in the results produced. It is important to be careful and not jump into decisions about which variant to use without considering the cost/benefit relationship between each choice, because a variant can produce better results, but be much more expensive to calculate, and therefore it is not viable to use such variant.

For the *Web* network, whose results can be seen in subfigures (e) there is also a considerable difference between both centrality measures, specially for the metrics outside of the giant connected component. For the metrics inside the giant connected component, EBC with recalculation produces results closer to the SEB centrality measure, similarly to the *C.elegans* network. The difference between using or not recalculation for each centrality measure is more impactful for the EBC centrality measure, since the differences are much more noticeable, than for the SEB variations. In fact, EBC with recalculation does not produce good results, certainly not better than EBC without recalculation, so the extra computational cost is not worth, for this specific network. The same logic can be extended to the SEB centrality measure, because if both variants produce similar values, for most metrics, it is not worth to choose the most computationally expensive.

# 6

## Conclusion

### Contents

---

6.1 Future Work . . . . .	67
---------------------------	----

---



This work focused on Spanning Edge Betweenness, a centrality measure related to edges. Firstly some concepts were introduced to better understand this field of study. We then proposed an implementation using *Python*. Also, an attempt to lower the computational time was conducted, as SEB can become quite expensive to calculate, especially for bigger networks. This attempt consists in taking advantage of the calculation being highly parallelizable or applying preprocessing in a way that would lower the calculation complexity. This attempt led to interesting results, which showed that it is possible to achieve lower calculation times if the right choices are taken, heavily dependent on the network's characteristics.

Another possibility of lowering the computational cost was thought to be by calculating the approximate measure and not the exact measure. This alternative showed to have a high threshold of when it does actually compensate regarding the computational cost. However, there are networks present in this work, which are not that big, where it compensates to use the approximate measure.

Finally, percolation was performed for unweighted networks. The results originated by Spanning Edge Betweenness are compared to those from Edge Betweenness Centrality. There are clear differences between the two centrality measures, as expected since they rely on different attributes and characteristics to assign each edge its value. It was also studied the impact of recalculating these values for both measures after each edge removal. The results show, for most metrics, a considerable difference between applying or not recalculation, especially for Edge Betweenness Centrality. If the difference makes up for the increase in calculation cost depends on the usage of such values.

## 6.1 Future Work

It is always possible to go deeper in every study. This one is no exception. Regarding Spanning Edge Betweenness implementation, there may be more options available to parallelize the implementation that were not taken into consideration. It is also possible that anytime after this work is published, some other option is created, which may achieve better results. The same goes for the preprocessing applied, where new and better ways may become available to use. It is also possible that some different preprocessing methods may be applied, either exclusively or together with the one already done.

For the weighted approximation, it is necessary a deeper analysis of the proposed implementation, where bigger and more networks are used for this purpose. It is also important to study generated networks and try to find relations between the calculation time and the characteristics of such networks. It is also possible to implement the full algorithm using the *Julia* programming language, which is expected to be much faster than the *Python* programming language, achieving better empirical results, even if the theoretical complexity stays the same.

For percolation, it would be interesting to include more edge-related measures in these types of

studies. However, these centrality measures are unavailable in network-related packages, making it almost impossible to integrate them. Hopefully, in the future, they do become available, and studies like this can include them. It is also important to extend these studies to weighted networks. However, to use Spanning Edge Betweenness in these studies, it first needs to be studied the different possible meanings of the different values that can be assigned to an edge, and taken into consideration when performing such studies.

# Bibliography

- [1] M. B. Baig and L. Akoglu, "Correlation of node importance measures: An empirical study through graph robustness," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15 Companion. New York, NY, USA: Association for Computing Machinery, 2015, p. 275–281.
- [2] Y. Qian, Y. Li, M. Zhang, G. Ma, and F. Lu, "Quantifying edge significance on maintaining global connectivity," *Scientific Reports*, vol. 7, no. 1, Mar. 2017.
- [3] A.-L. Barabási, "Network science," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 371, no. 1987, p. 20120375, 2013.
- [4] A. S. Teixeira, P. T. Monteiro, J. A. Carriço, M. Ramirez, and A. P. Francisco, "Spanning edge betweenness," in *Workshop on mining and learning with graphs*, vol. 24, 2013, pp. 27–31.
- [5] E. RENYI, "On random graph," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959. [Online]. Available: <https://ci.nii.ac.jp/naid/10026207309/en/>
- [6] E. N. Gilbert, "Random graphs," *The Annals of Mathematical Statistics*, vol. 30, no. 4, pp. 1141–1144, 1959. [Online]. Available: <http://www.jstor.org/stable/2237458>
- [7] J. Travers and S. Milgram, *An experimental study of the small world problem*. Princeton University Press, 2011, pp. 130–148.
- [8] S. Milgram, "The small world problem," *Psychology today*, vol. 2, no. 1, pp. 60–67, 1967.
- [9] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.
- [10] R. Albert, H. Jeong, and A.-L. Barabási, "Diameter of the world-wide web," *Nature*, vol. 401, no. 6749, pp. 130–131, Sep. 1999.
- [11] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

- [12] S. N. Dorogovtsev, J. F. F. Mendes, and A. N. Samukhin, "Size-dependent degree distribution of a scale-free growing network," *Phys. Rev. E*, vol. 63, p. 062101, May 2001.
- [13] P. Bonacich, "Power and centrality: A family of measures," *American Journal of Sociology*, vol. 92, no. 5, pp. 1170–1182, Mar. 1987.
- [14] P. Bonacich and P. Lloyd, "Eigenvector-like measures of centrality for asymmetric relations," *Social Networks*, vol. 23, no. 3, pp. 191–201, 2001.
- [15] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, Mar. 1953.
- [16] G. Sabidussi, "The centrality index of a graph," *Psychometrika*, vol. 31, no. 4, pp. 581–603, 1966.
- [17] E. Mones, L. Vicsek, and T. Vicsek, "Hierarchy measure for complex networks," *PLOS ONE*, vol. 7, no. 3, pp. 1–10, 03 2012.
- [18] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977. [Online]. Available: <http://www.jstor.org/stable/3033543>
- [19] U. Brandes, "A faster algorithm for betweenness centrality\*," *The Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163–177, Jun. 2001.
- [20] M. Piraveenan, M. Prokopenko, and L. Hossain, "Percolation centrality: Quantifying graph-theoretic impact of nodes during percolation in networks," *PloS one*, vol. 8, no. 1, p. e53095, Jan. 2013.
- [21] P. Boldi and S. Vigna, "Axioms for centrality," *Internet Mathematics*, vol. 10, no. 3-4, pp. 222–262, Jul. 2014.
- [22] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1, pp. 107–117, 1998, proceedings of the Seventh International World Wide Web Conference.
- [23] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.
- [24] P. Holme, B. J. Kim, C. N. Yoon, and S. K. Han, "Attack vulnerability of complex networks," *Phys. Rev. E*, vol. 65, p. 056109, May 2002.
- [25] W.-X. Wang and G. Chen, "Universal robustness characteristic of weighted networks against cascading failure," *Phys. Rev. E*, vol. 77, p. 026101, Feb 2008.

- [26] A. Barrat, M. Barthélemy, R. Pastor-Satorras, and A. Vespignani, "The architecture of complex weighted networks," *Proceedings of the National Academy of Sciences*, vol. 101, no. 11, pp. 3747–3752, 2004.
- [27] T. Alahakoon, R. Tripathi, N. Kourtellis, R. Simha, and A. Iamnitchi, "K-path centrality: A new centrality measure in social networks," in *Proceedings of the 4th Workshop on Social Network Systems*, ser. SNS '11. New York, NY, USA: Association for Computing Machinery, 2011.
- [28] R. Diestel, *Graph Theory*, ser. Electronic library of mathematics. Springer, 2006. [Online]. Available: <https://books.google.pt/books?id=aR2TMYQr2CMC>
- [29] P. Crucitti, V. Latora, M. Marchiori, and A. Rapisarda, "Error and attack tolerance of complex networks," *Physica A: Statistical mechanics and its applications*, vol. 340, no. 1-3, pp. 388–394, 2004.
- [30] C. Mavroforakis, R. Garcia-Lebron, I. Koutis, and E. Terzi, "Spanning edge centrality: Large-scale computation and applications," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2015, p. 732–742.
- [31] X.-Q. Cheng, F.-X. Ren, H.-W. Shen, Z.-K. Zhang, and T. Zhou, "Bridgeness: a local index on edge significance in maintaining global connectivity," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2010, no. 10, p. P10011, oct 2010.
- [32] M. Marchiori and V. Latora, "Harmony in the small-world," *Physica A: Statistical Mechanics and its Applications*, vol. 285, no. 3, pp. 539–546, 2000.
- [33] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.
- [34] M. Newman and D. Walls, *Scaling and percolation in the small-world network model*. Princeton University Press, 2011, pp. 310–320.
- [35] R. Albert, H. Jeong, and A.-L. Barabási, "Error and attack tolerance of complex networks," *nature*, vol. 406, no. 6794, pp. 378–382, 2000.
- [36] W. Ellens and R. E. Kooij, "Graph measures and network robustness," *CoRR*, vol. abs/1311.5064, 2013. [Online]. Available: <http://arxiv.org/abs/1311.5064>
- [37] A. Mercier, "Contagion-preserving network sparsifiers: Exploring epidemic edge importance utilizing effective resistance," *CoRR*, vol. abs/2101.11818, 2021. [Online]. Available: <https://arxiv.org/abs/2101.11818>

- [38] A. M. Mercier, S. V. Scarpino, and C. Moore, “Effective resistance for pandemics: Mobility network sparsification for high-fidelity epidemic simulation,” 2021.
- [39] R.-H. Li, J. X. Yu, X. Huang, H. Cheng, and Z. Shang, “Measuring robustness of complex networks under mvc attack,” in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, ser. CIKM '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1512–1516.
- [40] D. E. J. Wang, “Fast approximation of centrality,” *Graph algorithms and applications*, vol. 5, no. 5, p. 39, 2006.
- [41] U. Brandes and D. Fleischer, “Centrality measures based on current flow,” in *Annual symposium on theoretical aspects of computer science*. Springer, 2005, pp. 533–544.
- [42] D. A. Bader, S. Kintali, K. Madduri, and M. Mihail, “Approximating betweenness centrality,” in *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 2007, pp. 124–137.
- [43] R. Geisberger, P. Sanders, and D. Schultes, “Better approximation of betweenness centrality,” in *2008 Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, SIAM. USA: Society for Industrial and Applied Mathematics, 2008, pp. 90–100.
- [44] M. R. F. Mendonça, A. M. S. Barreto, and A. Ziviani, “Approximating network centrality measures using node embedding and machine learning,” *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 220–230, 2021.
- [45] F. Grando, L. Z. Granville, and L. C. Lamb, “Machine learning in network centrality measures: Tutorial and outlook,” *ACM Comput. Surv.*, vol. 51, no. 5, oct 2018. [Online]. Available: <https://doi.org/10.1145/3237192>
- [46] O. Wing and P. Demetriou, “Analysis of probabilistic networks,” *IEEE Transactions on Communication Technology*, vol. 12, no. 3, pp. 38–40, 1964.
- [47] S. E. Schaeffer, V. Valdés, J. Figols, I. Bachmann, F. Morales, and J. Bustos-Jiménez, “Characterization of robustness and resilience in graphs: a mini-review,” *Journal of Complex Networks*, vol. 9, no. 2, 05 2021, cnab018.
- [48] A. Sydney, C. M. Scoglio, P. Schumm, and R. E. Kooij, “ELASTICITY: topological characterization of robustness in complex networks,” *CoRR*, vol. abs/0811.4040, 2008. [Online]. Available: <http://arxiv.org/abs/0811.4040>
- [49] P. Van Mieghem, C. Doerr, H. Wang, J. M. Hernandez, D. Hutchison, M. Karaliopoulos, and R. Kooij, “A framework for computing topological network robustness,” *Delft University of Technology, Report20101218*, pp. 1–15, 2010.

- [50] A. H. Dekker and B. D. Colbert, "Network robustness and graph topology," in *Proceedings of the 27th Australasian Conference on Computer Science - Volume 26*, ser. ACSC '04. AUS: Australian Computer Society, Inc., 2004, p. 359–368.
- [51] V. Latora and M. Marchiori, "Efficient behavior of small-world networks," *Phys. Rev. Lett.*, vol. 87, p. 198701, Oct 2001.
- [52] W. Ellens, F. Spieksma, P. Van Mieghem, A. Jamakovic, and R. Kooij, "Effective graph resistance," *Linear Algebra and its Applications*, vol. 435, no. 10, pp. 2491–2506, 2011, special Issue in Honor of Dragos Cvetkovic.
- [53] T. Hayashi, T. Akiba, and Y. Yoshida, "Efficient algorithms for spanning tree centrality." in *IJCAI*, vol. 16, 2016, pp. 3733–3739.
- [54] G. Kirchhoff, "Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird," *Annalen der Physik*, vol. 148, no. 12, pp. 497–508, 1847.
- [55] D. Eppstein, "Representing all minimum spanning trees with applications to counting and generation," 1995.
- [56] J. G. Francis, "The qr transformation a unitary analogue to the lr transformation—part 1," *The Computer Journal*, vol. 4, no. 3, pp. 265–271, 1961.
- [57] V. N. Kublanovskaya, "On some algorithms for the solution of the complete eigenvalue problem," *USSR Computational Mathematics and Mathematical Physics*, vol. 1, no. 3, pp. 637–657, 1962.
- [58] A. Schwarzenberg-Czerny, "On matrix factorization and efficient least squares solution." *Astronomy and Astrophysics Supplement Series*, vol. 110, p. 405, 1995.
- [59] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, Í. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.
- [60] 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>
- [61] P. G. Doyle and J. L. Snell, *Random walks and electric networks*. American Mathematical Soc., 1984, vol. 22.
- [62] B. Bollobás, *Modern graph theory*. Springer Science & Business Media, 1998, vol. 184.
- [63] W. B. Johnson, "Extensions of lipschitz mappings into a hilbert space," *Contemp. Math.*, vol. 26, pp. 189–206, 1984.
- [64] D. A. Spielman and N. Srivastava, "Graph sparsification by effective resistances," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1913–1926, 2011. [Online]. Available: <https://doi.org/10.1137/080734029>
- [65] D. Achlioptas, "Database-friendly random projections," in *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 274–281. [Online]. Available: <https://doi.org/10.1145/375551.375608>
- [66] I. Koutis, G. L. Miller, and D. Tolliver, "Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing," *Computer Vision and Image Understanding*, vol. 115, no. 12, pp. 1638–1646, 2011, special issue on Optimization for Vision, Graphics and Medical Imaging: Theory and Applications. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1077314211001627>

