Spanning edge betweenness for large graphs and percolation

Guilherme Ribeiro
guilherme.ribeiro@tecnico.ulisboa.pt
Instituto Superior Técnico
Lisboa, Portugal

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.

CCS CONCEPTS

• **Networks**; • **Software and its engineering** → **Application specific development environments**; • **Theory of computation** → **Sparsification and spanners**;

KEYWORDS

Spanning Edge Betweenness, Approximate centrality measures, Edge percolation.

1 INTRODUCTION

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.

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" [Barabási 2013]. 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 [Teixeira et al. 2013]. 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.

1.2 Outline

In this work, are introduced basic concepts of Network Science, including graph definitions and representations while also presenting different network models. A brief explanation of Spanning Edge Betweenness can be found, together with efforts to reduce the computational cost of this centrality measure. Next there is the calculation of the approximation of Spanning Edge Betweenness for weighted graphs, along with results for such implementation. Finally there is a brief example of the percolation process and respective analysis.

2 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 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), or 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 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 $\in E$ 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. 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.

2.2 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.

2.2.1 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 [RENYI 1959]:

- $G(N, L)$ 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 [Gilbert 1959]:

- $G(N, p)$ 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.

2.2.2 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 [Travers and Milgram 2011]. 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 [Milgram 1967]. 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 [Watts and Strogatz 1998]:

- 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.

2.2.3 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 [Albert et al. 1999], 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 [Albert et al. 1999], defined as:

$$p_k = k^{-\gamma} \tag{1}$$

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* [Barabási and Albert 1999]:

- 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* [Dorogovtsev et al. 2001]:

- 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 [Dorogovtsev et al. 2001], 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*.

2.3 Centrality Measures

This section is divided into two subsections. First, we present the centrality measures, and how to compute them, regarding nodes and then, the same but with centrality measures related to the 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.

2.3.1 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:

$$E(e) = \sum_{i,j \in V} \frac{e \in \sigma(i,j)}{\sigma(i,j)}, \tag{2}$$

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 [Girvan and Newman 2002].

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:

$$Dp(e) = D(i)D(j) \tag{3}$$

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 [Barrat et al. 2004; Holme et al. 2002; Wang and Chen 2008].

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:

$$B(e) = \frac{\sqrt{S_i S_j}}{S_e}, \tag{4}$$

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:

$$K(e) = \sum_{i \in V} \frac{e \in \sigma_k(i)}{\sigma_k(i)}, \tag{5}$$

where $\sigma_k(i)$ represents a path with length k from node i. The notion of K-Path centrality was introduced in [Alahakoon et al. 2011].

Spanning Edge Betweenness. unlike other measures already mentioned, the Spanning Edge Betweenness centrality measure [Teixeira et al. 2013], 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:

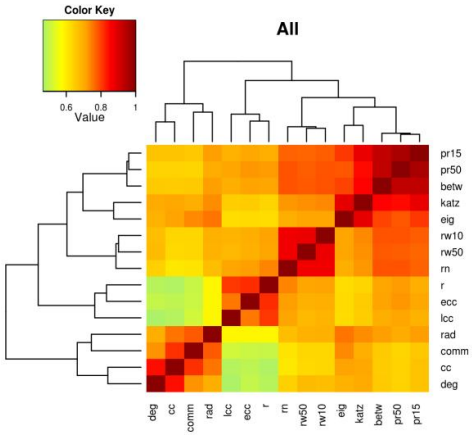


Figure 1: Image retrieved from [Baig and Akoglu 2015], illustrating correlation between different studied centrality measures. A color scale was used, where green represents less similarity and red represents more similarity. The three similarity clusters are: betw-pr15, katz-eig and deg-cc.

$$S(e) = \frac{e \in \tau_G}{\tau_G}, \tag{6}$$

where $e \in \tau_G$ represents the number of MSTs that contain edge e, and τ_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.

2.4 Related work

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 or small real-world networks. 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 [Mavroforakis et al. 2015], 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 centrality 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 [Baig and Akoglu 2015], 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.

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 [Brandes and Fleischer 2005; Wang 2006], 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 [Bader et al. 2007], 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 [Geisberger et al. 2008], 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 tat they can achieve results that are very close to the exact ones only using a fraction of the time [Grando et al. 2018; Mendonça et al. 2021].

3 SPANNING EDGE BETWEENNESS – IMPLEMENTATION AND PARALLELISM

In this section we briefly explain the theory behind Spanning Edge Betweenness, and based on that we propose an implementation, together with a couple of possible improvements in order to reduce the computational time cost.

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 [Teixeira et al. 2013], but quickly other applications started to emerge, one of them being the study of network robustness.

SEB can be formally defined as:

$$SEB(e) = \frac{e \in \tau_G}{\tau_G}, \tag{7}$$

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 τ_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 [Kirchhoff 1847] for unweighted networks and on Eppstein [Eppstein 1995] for weighted networks.

In [Teixeira et al. 2013] is explained how to compute both $e \in \tau_G$ and τ_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.

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 [Francis 1961] and Vera N. Kublanovskaya [Kublanovskaya 1962], independently. The *LU decomposition* was introduced by Tadeusz Banachiewicz in 1938 [Schwarzenberg-Czerny 1995]. 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 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.

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.

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*[1] [Virtanen et al. 2020] and *NumPy*[2] [Harris et al. 2020] 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 possibilitates the easiest method to calculate the determinant of a matrix, hence it is the one that will be used.

[1] https://scipy.org/
[2] https://numpy.org/

```
begin
    edges ⟵ sortEdgesByWeight(G.edges())
    Laplacian ⟵ laplacianMatrix(G)
    LaplacianToKirchhoff ⟵
     RemoveRowColumn(Laplacian, 0, 1)
    TotalMSTs ⟵
     CalculateDeterminant(LaplacianToKirchhoff)
    n ⟵ 0
    while True do
        if StillInTheSameWeightValue() then
            AddEdgeToLaplacian(edges[n])
            n ⟵ n + 1
        end
        if AnyEdgeProcessed() then
            BuildSCCs()
            CalculateSEBEachSCC()
            if NoMoreEdges() then
                break
            end
            ContractEdgesEachSCC()
        end
        else
            IncreaseWeightValue()
        end
    end
end
```

Algorithm 1: Weighted Network

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 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.

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.

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.

Below there is a brief explanation for every real world network, used for the studies which results are present in this work.

- **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.

In Table 1 there are the characteristics for the real world networks used in this study. It is possible to see that the networks are very heterogeneous, not only in size but also in terms of clustering coefficient, assortativity and average degree.

Table 1: Characteristics of real world networks. C = Clustering Coefficient, r = assortativity, <k> = average degree.

Network	# Nodes	# Edges	C	r	<k>
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

Table 2: Real world networks SEB calculation time without preprocessing, with preprocessing and using PySpark, time in seconds.

Network	# Edges	# bridges	Without	With	PySpark
PowerGrid	6594	1611	7936	3126	26830
Facebook	2981	2796	1502	0.8	10109
Dolphins	159	9	0.60	0.35	21
C.elegans	2025	8	307	212	3016
NetScience	914	30	78	63	1171
Web	6474	512	3696	1134	22306

In Table 2 there is the computation time for SEB in each real world network used. The first two columns show the characteristics for each network, regarding the number of edges and bridges. The third columns shows the time when neither preprocessing nor parallelization is applied. The fourth column shows the time when only preprocessing is applied. Finally, the last column shows the time when only parallelization, using PySpark, is applied.

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 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. This was the preprocessing performed in this work.

There is a clear improvement from the preprocessing and an even more clear negative effect on the calculation time when using PySpark.

4 WEIGHTED APPROXIMATION

In the previous sections 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 [Mavroforakis et al. 2015] for unweighted networks. The authors took advantage of the relation between SEB and the effective resistance concept [Doyle and Snell 1984]. 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 [Bollobás 1998; Doyle and Snell 1984], therefore $R(e) = SEB(e)$. Effective resistance can be seen as pairwise distances between vectors [Mavroforakis et al. 2015]. This allows us to use the Johnson-Lindenstraus Lemma [Johnson 1984], 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 2, that was first proposed by Spielman and Srivastava [Spielman and Srivastava 2011].

begin
  $Z \longleftarrow [], L \longleftarrow laplacianMatrix(G), B \longleftarrow IncidenceMatrix(G)$
  $Q \longleftarrow RandomProjectionMatrix(k, m)$
  $Y = QB$
  for *i=1 ... k* **do**
    Solve $Lz_i = Y[i, :]$
    $Z = [Z; z_i^T]$
  end
  For every e=(u,v), return $R(e) = \|Z[u, :] - Z[v, :]\|_2^2$
end

Algorithm 2: SEB approximations for unweighted networks.

By looking at Algorithm 2 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 from [Achlioptas 2001], Theorem 2. In this case the second option was chosen.

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

$$k_0 = \frac{4 + 2\beta}{\epsilon^2/2 - \epsilon^3/3} log(n) \tag{8}$$

where β controls the probability of success and ϵ 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 $u, v \in E$

$$(1 - \epsilon)R(e) \leq ER(e) \leq (1 + \epsilon)R(e) \tag{9}$$

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 8. 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 ϵ and β, 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 2, 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 [Mavroforakis et al. 2015] 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 [Koutis et al. 2011]. 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 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 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.

4.1 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 3, 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 theirs properties can be consulted in Table 1.

Table 3: 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 3 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[3] 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

[3]https://github.com/danspielman/Laplacians.jl/blob/master/docs/src/usingSolvers.md

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 3 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 1 where it is possible to observe for each network the number of nodes and 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 ϵ and β, 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 *PowerGrid* 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 PERCOLATION

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.

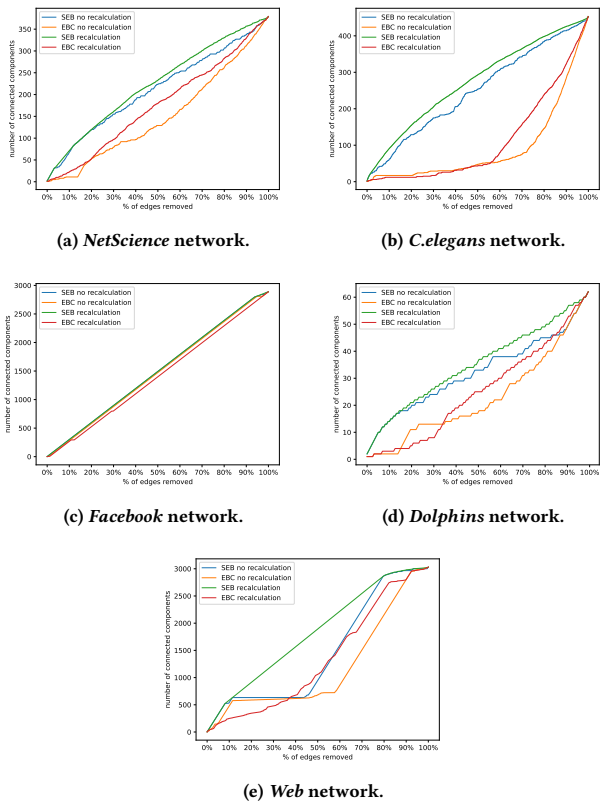
In this section edge percolation was performed on real world networks. It is important to recall the networks' characteristics, that can be seen in Table 1. The aim of this chapter is to compare SEB to Edge Betweenness Centrality(EBC) when percolation is performed on a network based on the value that 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.

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 1. The results for the *NetScience* network are represented in subfigure (a). 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 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 subfigure (c), it shows 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. Subfigure (d) shows 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 subfigure (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.

In Figure 2 there is an example of the percolation performed. In the yy axis is the number of connected components, and in the xx axis is the percentage of edges removed. By looking at this same figure it is possible to draw some conclusions. For about every network there is a clear difference between Spanning Edge Betweenness(SEB) and Edge Betweenness Centrality(EBC), where for this specific measure, SEB achieves higher number of connected components. There is also a noticeable difference between recalculating or not the centrality values, for both measures, that is more obvious for EBC than for SEB. However, it can be argued that the difference is not big enough to justify the increase in computational time introduced by the recalculation of the centrality values.



(a) *NetScience* network. **(b) *C.elegans* network.**

(c) *Facebook* network. **(d) *Dolphins* network.**

(e) *Web* network.

Figure 2: Evolution of the number of connected components with the removal of edges, for every real world network studied.

6 CONCLUSION

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.

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.

ACKNOWLEDGMENTS

To Prof. Pedro Monteiro, Prof. Sofia Teixeira, thanks for everything. To Prof. Alexandre, thanks for all the help and availability to answer my questions. To my family, friends and girlfriend, without you this would not be possible.

REFERENCES

Dimitris Achlioptas. 2001. Database-Friendly Random Projections. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Santa Barbara, California, USA) *(PODS '01)*. Association for Computing Machinery, New York, NY, USA, 274–281. https://doi.org/10.1145/375551.375608

Tharaka Alahakoon, Rahul Tripathi, Nicolas Kourtellis, Ramanuja Simha, and Adriana Iamnitchi. 2011. K-Path Centrality: A New Centrality Measure in Social Networks. In *Proceedings of the 4th Workshop on Social Network Systems* (Salzburg, Austria)

(SNS '11). Association for Computing Machinery, New York, NY, USA, Article 1, 6 pages. https://doi.org/10.1145/1989656.1989657

Réka Albert, Hawoong Jeong, and Albert-László Barabási. 1999. Diameter of the World-Wide Web. *Nature* 401, 6749 (Sept. 1999), 130–131. https://doi.org/10.1038/43601

David A. Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. 2007. Approximating Betweenness Centrality. In *Algorithms and Models for the Web-Graph*, Anthony Bonato and Fan R. K. Chung (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 124–137.

Mirza Basim Baig and Leman Akoglu. 2015. Correlation of Node Importance Measures: An Empirical Study through Graph Robustness. In *Proceedings of the 24th International Conference on World Wide Web* (Florence, Italy) *(WWW '15 Companion)*. Association for Computing Machinery, New York, NY, USA, 275–281. https://doi.org/10.1145/2740908.2743055

Albert-László Barabási. 2013. Network science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 371, 1987 (2013), 20120375. https://doi.org/10.1098/rsta.2012.0375

Albert-László Barabási and Réka Albert. 1999. Emergence of Scaling in Random Networks. *Science* 286, 5439 (1999), 509–512. https://doi.org/10.1126/science.286.5439.509 arXiv:https://www.science.org/doi/pdf/10.1126/science.286.5439.509

A. Barrat, M. Barthélemy, R. Pastor-Satorras, and A. Vespignani. 2004. The architecture of complex weighted networks. *Proceedings of the National Academy of Sciences* 101, 11 (2004), 3747–3752. https://doi.org/10.1073/pnas.0400087101 arXiv:https://www.pnas.org/content/101/11/3747.full.pdf

Béla Bollobás. 1998. *Modern graph theory*. Vol. 184. Springer Science & Business Media.

Ulrik Brandes and Daniel Fleischer. 2005. Centrality Measures Based on Current Flow. In *STACS 2005*, Volker Diekert and Bruno Durand (Eds.). Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 533–544.

S. N. Dorogovtsev, J. F. F. Mendes, and A. N. Samukhin. 2001. Size-dependent degree distribution of a scale-free growing network. *Phys. Rev. E* 63 (May 2001), 062101. Issue 6. https://doi.org/10.1103/PhysRevE.63.062101

Peter G Doyle and J Laurie Snell. 1984. *Random walks and electric networks*. Vol. 22. American Mathematical Soc.

David Eppstein. 1995. Representing all minimum spanning trees with applications to counting and generation. (1995).

John GF Francis. 1961. The QR transformation a unitary analogue to the LR transformation—Part 1. *Comput. J.* 4, 3 (1961), 265–271.

Robert Geisberger, Peter Sanders, and Dominik Schultes. 2008. Better approximation of betweenness centrality. In *2008 Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, Society for Industrial and Applied Mathematics, USA, 90–100.

E. N. Gilbert. 1959. Random Graphs. *The Annals of Mathematical Statistics* 30, 4 (1959), 1141–1144. http://www.jstor.org/stable/2237458

M. Girvan and M. E. J. Newman. 2002. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences* 99, 12 (2002), 7821–7826. https://doi.org/10.1073/pnas.122653799 arXiv:https://www.pnas.org/content/99/12/7821.full.pdf

Felipe Grando, Lisandro Z. Granville, and Luis C. Lamb. 2018. Machine Learning in Network Centrality Measures: Tutorial and Outlook. *ACM Comput. Surv.* 51, 5, Article 102 (oct 2018), 32 pages. https://doi.org/10.1145/3237192

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. 2002. Attack vulnerability of complex networks. *Phys. Rev. E* 65 (May 2002), 056109. Issue 5. https://doi.org/10.1103/PhysRevE.65.056109

William B Johnson. 1984. Extensions of Lipschitz mappings into a Hilbert space. *Contemp. Math.* 26 (1984), 189–206.

Gustav Kirchhoff. 1847. Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird. *Annalen der Physik* 148, 12 (1847), 497–508.

Ioannis Koutis, Gary L. Miller, and David Tolliver. 2011. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Computer Vision and Image Understanding* 115, 12 (2011), 1638–1646. https://doi.org/10.1016/j.cviu.2011.05.013 Special issue on Optimization for Vision, Graphics and Medical Imaging: Theory and Applications.

Vera N Kublanovskaya. 1962. On some algorithms for the solution of the complete eigenvalue problem. *U. S. S. R. Comput. Math. and Math. Phys.* 1, 3 (1962), 637–657.

Charalampos Mavroforakis, Richard Garcia-Lebron, Ioannis Koutis, and Evimaria Terzi. 2015. Spanning Edge Centrality: Large-Scale Computation and Applications. In *Proceedings of the 24th International Conference on World Wide Web* (Florence, Italy) *(WWW '15)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 732–742. https://doi.org/10.1145/2736277.2741125

Matheus R. F. Mendonça, André M. S. Barreto, and Artur Ziviani. 2021. Approximating Network Centrality Measures Using Node Embedding and Machine Learning. *IEEE Transactions on Network Science and Engineering* 8, 1 (2021), 220–230. https://doi.org/10.1109/TNSE.2020.3035352

Stanley Milgram. 1967. The small world problem. *Psychology today* 2, 1 (1967), 60–67.

Erdos RENYI. 1959. On Random Graph. *Publicationes Mathematicate* 6 (1959), 290–297. https://ci.nii.ac.jp/naid/10026207309/en/

Alex Schwarzenberg-Czerny. 1995. On matrix factorization and efficient least squares solution. *Astronomy and Astrophysics Supplement Series* 110 (1995), 405.

Daniel A. Spielman and Nikhil Srivastava. 2011. Graph Sparsification by Effective Resistances. *SIAM J. Comput.* 40, 6 (2011), 1913–1926. https://doi.org/10.1137/080734029 arXiv:https://doi.org/10.1137/080734029

Andreia Sofia Teixeira, Pedro T Monteiro, João A Carriço, Mário Ramirez, and Alexandre P Francisco. 2013. Spanning edge betweenness. In *Workshop on mining and learning with graphs*, Vol. 24. Springer International Publishing, Cham, 27–31.

Jeffrey Travers and Stanley Milgram. 2011. *An experimental study of the small world problem*. Princeton University Press, 130–148. https://doi.org/10.1016/B978-0-12-442450-0.50018-3

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

David Eppstein Joseph Wang. 2006. Fast approximation of centrality. *Graph algorithms and applications* 5, 5 (2006), 39.

Wen-Xu Wang and Guanrong Chen. 2008. Universal robustness characteristic of weighted networks against cascading failure. *Phys. Rev. E* 77 (Feb 2008), 026101. Issue 2. https://doi.org/10.1103/PhysRevE.77.026101

Duncan J. Watts and Steven H. Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature* 393, 6684 (June 1998), 440–442. https://doi.org/10.1038/30918