A Continuous-time ODE Framework to Centrality in Dynamic Networks

Francisco García Atienza

Bachelor's thesis
2023:K10

Lund University

Faculty of Science
Centre for Mathematical Sciences
Numerical Analysis

Centrum Scientiarum Mathematicarum

García Atienza, Francisco:

A Continuous-time ODE Framework to Centrality in Dynamic Networks

Bacherlor's Thesis, Centre for Mathematical Sciences, Lund University, 2023.

Abstract

The present study discusses and evaluates a novel continuous-time approach to network centrality, a fundamental concept in network analysis. Static methods based on discrete snapshots or time slices for measuring node interaction in networks that change over time do not fully capture the dynamical nature of network evolution. To address this issue, a dynamical system model driven by the continuous-time adjacency matrix of the network is derived from a continuous version of Katz centrality, one of the most widely used centrality measures. Using this approach, numerical experiments are conducted on various networks, synthetic and real, showing that this continuous-time framework performs better than traditional static or aggregate measures and that it is able to capture dynamic effects of communication between nodes or even changes in the network structure over time.

The most significant conclusions of this work are that the new ODE-based framework offers major improvements in accuracy and efficiency over static methods for network simulations. By using advanced numerical ODE solvers, time discretization is performed automatically "under the hood" in an optimal and efficient manner, allowing the system to adapt to sudden and significant changes in network behavior. The ODE system's property of downweighting information over time also enables real-time monitoring of centrality rankings without the need to store or account for all previous node interaction history. Moreover, it is shown why tracking good receivers of information in a dynamic network is cheaper than tracking good broadcasters from a computational cost perspective.

Overall, this dynamical systems approach to network centrality can lead to new insights into the behavior of complex networks and has implications for a wide range of applications, from social networks to fluid mechanics, where network dynamics play a crucial role.

Acknowledgements

With this Bachelor's degree project, a fascinating journey ends and a new one begins. I would like to thank all the teachers and classmates I have had throughout these three years. From each and every one of them I have learned, and am still learning, to be a better mathematician.

Special thanks to my examiner Philipp Birken, for presenting me with a very interesting thesis subject, and to my supervisor, Viktor Linders, for his invaluable advice and guidance from the beginning of this project.

Thanks to my family for their unconditional love.

Finally, and above all, this thesis work is dedicated to my lovely husband, Alfredo, who has been a constant source of love and support. Thanks for believing in me more than myself and teaching me that with humility, work and effort, dreams can be achieved in life.

Lund, May 2023 *Francisco García Atienza*

Contents

Chapter 1

Introduction

Networks are fundamental in many fields of science as they provide a powerful way to model complex systems and relationships between entities at different scales. By representing entities as nodes and connections between them as edges, networks can help researchers understand how information, energy, materials, and other resources flow through a system, identify patterns and structures within the data, and make predictions about system behavior. Many real life problems can be modeled as graphs or discretized as networks and that is why they are widely used in many fields such as physics, biology, sociology, statistics, or computer science among others. Network analysis in all these areas have led to significant advances in our understanding of the world around us [1], [2], [3, Part I].

1.1 Mathematical notation

This section introduces a brief review of the definitions and notation pertaining to graphs, as well as some of their matrix representations, that will be used later in this study.

A network in the mathematical literature is, in its simplest form, a collection of interconnected nodes or vertices that represent entities, and the connections or edges between these nodes that represent relationships or interactions between the entities [4].

Definition 1.1.1 *A weighted network, or graph, is an ordered triple of sets* $G = (V, E, \mathcal{W})$*, where* V *is the set of nodes,* $E \subset V \times V$ *is the set of edges among the nodes and* $\mathcal{W}$ *is a map that assigns to each edge a weight or cost, usually a positive real number. When all these weights are given the same relevance or weight (by convention,* $\omega = 1$*), then the graph, represented only by* $G = (V, E)$*, is called unweighted.*

The following two definitions that are central for this work are the concept of *walk* and the *adjacency matrix* of a graph.

Definition 1.1.2 *A walk of length w is a sequence of w edges $(e_1, e_2, \ldots, e_w)$ such that the target of e_ℓ coincides with the source of $e_{\ell+1}$ for all $\ell = 1, 2, ..., w-1$.*

Definition 1.1.3 *Let $G = (V, E, \mathcal{W})$ be a weighted graph with N nodes. Its adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ is entry-wise defined as*

$$\mathbf{A}_{ij} = \begin{cases} \sum_{e_{ij}} \omega_{ij} & \textit{if there is an edge } e_{ij} \textit{ between nodes } i \textit{ and } j \textit{ with cost } \omega_{ij}, \\ 0 & \textit{otherwise,} \end{cases} \tag{1.1}$$

for all $i, j = 1, 2, \ldots, N$.

Unless otherwise stated, all graphs analyzed in this study will be considered *simple networks*.

Definition 1.1.4 *A simple network is a type of network where each edge connects only two distinct nodes, i.e., there are no self-loops or multiple edges between the same pair of nodes.*

As a particular case, the adjacency matrix for simple unweighted graphs will have all their non-zero entries set to $\mathbf{A}_{ij} = 1$, indicating a link between nodes i and j.

Additionally, graphs are usually represented visually using a diagram (e.g., Fig. 1.1), where nodes are represented as points and edges are represented as lines connecting the points. The direction of the edges allows us to divide graphs into *directed* and *undirected*. As a consequence of this, undirected graphs are represented by symmetric adjacency matrices, i.e., $\mathbf{A}_{ij} = \mathbf{A}_{ji}$ and, on the contrary, directed graphs by asymmetric matrices, $\mathbf{A}_{ij} \neq \mathbf{A}_{ji}$. The presence of loops, i.e., edges connecting a node to itself can also be indicated in this visual representation, $\mathbf{A}_{ii} = 1$, in terms of the adjacency matrix.

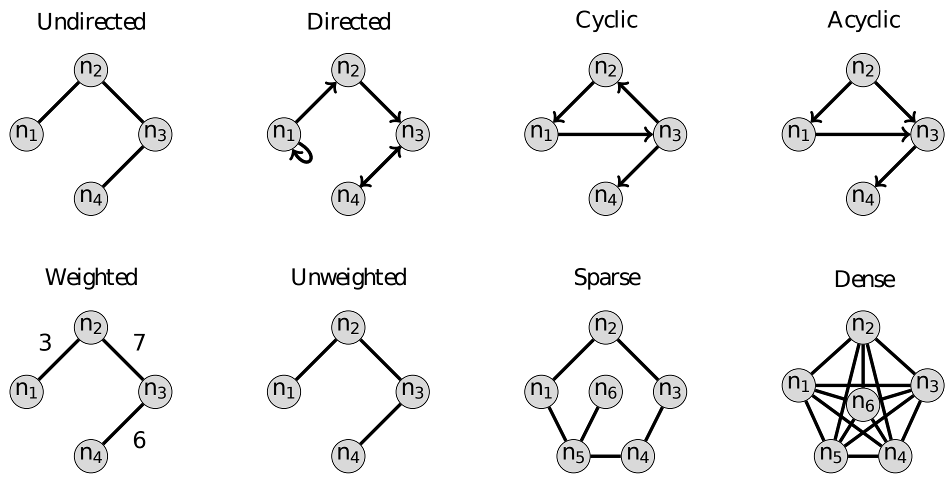


Figure 1.1: Examples of different network properties.

It is also worth mentioning other relevant network properties of graph theory for this work as:

- *Cyclic/Acyclic networks*: Cyclic networks are networks where there is at least one path from a node to itself. In other words, they contain cycles or loops. Examples of cyclic networks include recurrent neural networks (RNNs) and feedback loops in control systems. On the other hand, acyclic networks are networks that do not contain any cycles so there is no path from any node back to itself. Examples of these networks are directed acyclic graphs (DAGs) as directed binary trees.

- *Sparse/Dense networks*: respectively, a sparse/dense network is a type of network where the number of edges (or connections) between nodes is/is not much smaller than the maximum possible number of edges in the network, resulting in a relatively low/high density of connections. Although there is a widespread agreement that most empirical networks are sparse, there is no formal definition of sparsity for any finite network, being the notion of "much smaller" purely colloquial.

- *Static/Dynamic networks*: A static network is a network that does not change over time. The relationships and connections between nodes are fixed and remain constant. On the contrary, a dynamic network is a network that changes over time. The relationships and connections between nodes can evolve and alter, resulting in a continually changing network structure.

Linear algebra will play a significant role in network analysis as graphs are represented by matrices. The analysis of networks often involves solving linear systems, determining eigenvalues and eigenvectors, and evaluating matrix functions. Moreover, the examination of dynamic processes on graphs will create systems of differential equations based on their structure. The behavior of the solution over time is expected to be highly impacted by the graph's structure (network topology), which is reflected in the spectral properties of the matrices related to the graph. This turns out to be one of the most basic questions about the network's structure; the identification of most relevant nodes within a network. This leads us to the concept of *centrality*.

1.2 Centrality measures

Centrality measures are metrics that are used to quantify the relative importance/influence/position of a node in a network. Indicators of centrality assign numbers or rankings, usually the higher the more important, to nodes within a graph corresponding to their network position based on different criteria. This gives rise to several types of centrality measures [3, Ch. 7]:

Degree centrality

The Degree centrality measures the number of connections a node i has to other nodes in the network. This is called the degree of a node (k). Let $\mathbf{A} \in \mathbb{R}^{N\times N}$ be the adjacency matrix of a graph with N nodes. If we define $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ as the centrality vector of the graph, then we can express the Degree centrality for each node as

$$x_i^{(deg)} = k_i = \sum_{j=1}^{N} \mathbf{A}_{ij}, \quad i = 1, \ldots, N. \tag{1.2}$$

This measure is usually normalized by the maximal possible degree, N–1, to obtain a number between 0 and 1. For certain networks, Degree centrality can be very illuminating as it provides a straightforward and simple indication of a node's connectedness or level of popularity, but fails to consider other crucial elements of the network structure such as the importance of a node or its place within the network.

Closeness centrality

In order to extend the basic measure of degree and take into account the position of the nodes in the network, *closeness* and *betweenness* measures are defined. For its part, Closeness centrality measures the average distance, $\sum_j d(i,j)$, between a node and all other nodes in the network, where $d(i,j)$ is defined as the shortest path between nodes i and j, i.e, the minimum sum of the weights along the path between them. In its normalized version it can be expressed as

$$x_i^{(clos)} = \frac{N-1}{\sum_{j\neq i} d(i,j)}. \tag{1.3}$$

An alternative measure of Closeness centrality is the *Harmonic centrality* which aggregates distances differently as the sum of all inverses of distances, $\sum_j 1/d(i,j)$. This avoids having a few nodes for which there is a large or infinite distance,

$$x_i^{(har)} = \frac{1}{N-1} \sum_{j\neq i} \frac{1}{d(i,j)}. \tag{1.4}$$

Betweenness centrality

Betweenness centrality measures the number of times a node acts as a bridge along the shortest path between two other nodes in the network. Formally, if we redefine g_{jk}^i to be the number of shortest paths from j to k that pass through i and we define g_{jk} to be the total number of shortest paths from j to k, then the Betweenness centrality of node i on a general network is defined as

$$x_i^{(bet)} = \sum_{j<k} \frac{g_{jk}^i}{g_{jk}}. \tag{1.5}$$

Eigenvector centrality

The Eigenvector centrality measures the influence of a node based on the influence of its neighbors. Unlike Degree centrality which assigns one point for each network connection, Eigenvector centrality assigns points based on the centrality scores of a node's neighbors, resulting in a more nuanced understanding of a node's centrality. If we denote the centrality of node i by x_i where $\mathbf{x}$ is the centrality vector and $\mathbf{A}$ the adjacency matrix of the graph, then making use of the adjacency matrix and making x_i proportional to the average of the centralities of i's network neighbours we have

$$x_i = \kappa \sum_{j=1}^{N} \mathbf{A}_{ij} x_j, \tag{1.6}$$

where κ is a constant. We can rewrite this equation in matrix form considering $\kappa = 1/\lambda$ as

$$\lambda \mathbf{x} = \mathbf{A}\mathbf{x}. \tag{1.7}$$

Hence, $\mathbf{x}$ is the eigenvector of the adjacency matrix corresponding to the eigenvalue λ. By the Perron–Frobenius theorem [5, Ch. 8], a real square matrix with all elements non-negative, like an adjacency matrix, has only one eigenvector with all elements with the same sign and that is precisely the leading eigenvector. Therefore, assuming that we wish the centralities to be non-negative, it is shown that λ corresponds the spectral radius of $\mathbf{A}$, i.e., the largest absolute value of its eigenvalues, ($\kappa = (|\lambda_1|^{-1})$ and the centrality vector, $\mathbf{x}$, to its corresponding eigenvector.

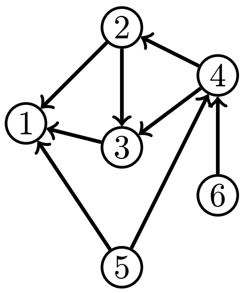


Figure 1.2: Acyclic graph.

Drawbacks of this type of measure are that it does not scale well for directed networks (asymmetric adjacency matrices) where it is not always possible to compute a unique, real leading eigenvector. Or, even worse, it is not applicable in acyclic networks. Fig. 1.2 illustrates how node 1 in that network has only in-going edges and hence will have eigenvector zero centrality, according to (1.6). Node 3 has an out-going edge and two in-going edges, but the in-going one is incident on node 1, and hence node 3 will also have zero centrality. Following this argument for the remainder nodes, the result is a zero centrality for the whole network. To address these problems, variants based on Eigenvector Centrality such as PageRank and Katz centrality were developed.

PageRank and Katz centrality

In the task of finding the most important nodes in a network, two of the most widely used methods are PageRank and Katz centrality. To solve Eigenvector centrality problems in networks that do not have strongly connected components of more than one node resulting in a zero centrality vector, the main idea of these two methods is to give each node a small amount of centrality for free.

Let $\mathbf{A} \in \mathbb{R}^{N\times N}$ be the adjacency matrix for an unweighted ($\mathbf{A}_{ij} = 1$), directed (with possible undirected links, $\mathbf{A}_{ij} = \mathbf{A}_{ji}$), static network of N nodes. We need the networks to be unweighted since, as we will see later, the $(ij)^{th}$ entry of the matrix $\mathbf{A}^k$ gives us a combinatorial way of counting paths of length k from i to j which will be essential to define the concept of centrality based on counting all possible paths from a node to all its neighbors in the network. For weighted graphs, this interpretation of the $(ij)^{th}$ entry of the matrix $\mathbf{A}^k$ has to be modified. Then, from (1.6) PageRank centrality is defined as

$$x_i = \alpha \sum_{j=1}^{N} \mathbf{A}_{ij} \frac{x_j}{k_j^{\text{out}}} + \beta, \tag{1.8}$$

where α, β are positive parameters and k_j^{out} denotes the out-degree of node j, i.e., the number of out-going links from that node. The first term correspond to the Eigenvector centrality and the second term is the “free” part, the constant extra amount that all nodes receive. By including this additional component, we make sure that nodes with no incoming connections still receive centrality, and once they have a non-zero centrality score, they can distribute it to the other nodes they are linked to. This results in nodes that are connected to many others having a high centrality, regardless of whether they are part of a strongly connected component or an out-component.

The difference between PageRank and Katz centrality resides precisely in the way they spread the centrality over the network. In mathematical terms, Katz centrality is obtained from (1.8) when $k_j^{\text{out}} = 1$ for each j,

$$x_i = \alpha \sum_{j=1}^{N} \mathbf{A}_{ij} x_j + \beta. \tag{1.9}$$

If a node with a high Katz score has links to many other nodes, then all of those linked nodes will also receive a high centrality score. PageRank, instead, is a variant in which the centrality derived from network neighbors is proportional to their centrality divided by their out-degree.

Therefore, nodes that point to many others pass only a small amount of centrality on to each of those others, even if their own centrality is high.

By convenience, we set $k_j^{\text{out}} = 1$ in (1.8) to avoid zero-division for nodes with no outgoing edges. Thus, we can express PageRank centrality in matrix form as

$$\mathbf{x} = \alpha \mathbf{A}\mathbf{D}^{-1}\mathbf{x} + \beta \mathbf{1}, \tag{1.10}$$

with $\mathbf{1}$ being the vector of ones $(1, \ldots, 1)$ and $\mathbf{D}$ being the diagonal matrix with elements $\mathbf{D}_{ii} = max(k_i^{\text{out}}, 1)$. Rearranging for $\mathbf{x}$ and setting the conventional value of $\beta = 1$, the PageRank centrality yields

$$\mathbf{x} = (\mathbf{I} - \alpha \mathbf{A}\mathbf{D}^{-1})^{-1}\mathbf{1}. \tag{1.11}$$

Again, a similar expression for Katz centrality is obtained if we consider $\mathbf{D}^{-1} = \mathbf{I}$, for $\mathbf{I}$ denoting the identity matrix of order N.

We seek α such that $\mathbf{I} - \alpha\mathbf{A}\mathbf{D}^{-1}$ is non-singular, i.e. $\det(\mathbf{I} - \alpha\mathbf{A}\mathbf{D}^{-1}) \neq 0$, or what is the same, $\det(\mathbf{A}\mathbf{D}^{-1} - \alpha^{-1}\mathbf{I}) \neq 0$. This it is simply the characteristic equation of $\mathbf{A}\mathbf{D}^{-1}$ whose roots are equal to the eigenvalues of the matrix $\mathbf{A}\mathbf{D}^{-1}$, $\lambda = \alpha^{-1}$. This suggests a good value for α bounded by $0 < \alpha < 1/\rho(\mathbf{A}\mathbf{D}^{-1})$, denoting $\rho(\cdot)$ the spectral radius (e.g. Google uses $\alpha = 0.85$ [6]).

In the choice of α we must take into account that the closer we are to the spectral radius, the maximum amount of weight on the eigenvector term will be place and the smallest amount on the constant term. If we let instead $\alpha \to 0$, then only the constant term will survive in (1.9) resulting in all nodes with equal centrality.

PageRank was developed by Google co-founders Larry Page and Sergey Brin as a way to rank websites in their search engine results. The basic idea behind PageRank is that a node is considered important if it is linked to by many other important nodes. The PageRank score of a node is determined by the sum of the PageRank scores of the nodes that link to it, with a damping factor applied to reduce the influence of nodes with many outbound links.

PageRank and Katz centrality are widely used in the field of network analysis and has been applied to a wide range of networks, including the World Wide Web, social networks, or biological networks [6], [7],[8, Ch. 5]. Overall, each centrality measure provides a different perspective on the importance of a node in a network (see Fig. 1.3) and can be useful in various applications, such as network analysis, recommendation systems, or identifying key players in complex systems. The most appropriate centrality measure will require a more detailed analysis of the specific characteristics of the network in question.

Katz centrality is discussed further in the next section 1.3 as it is the central topic of this thesis.

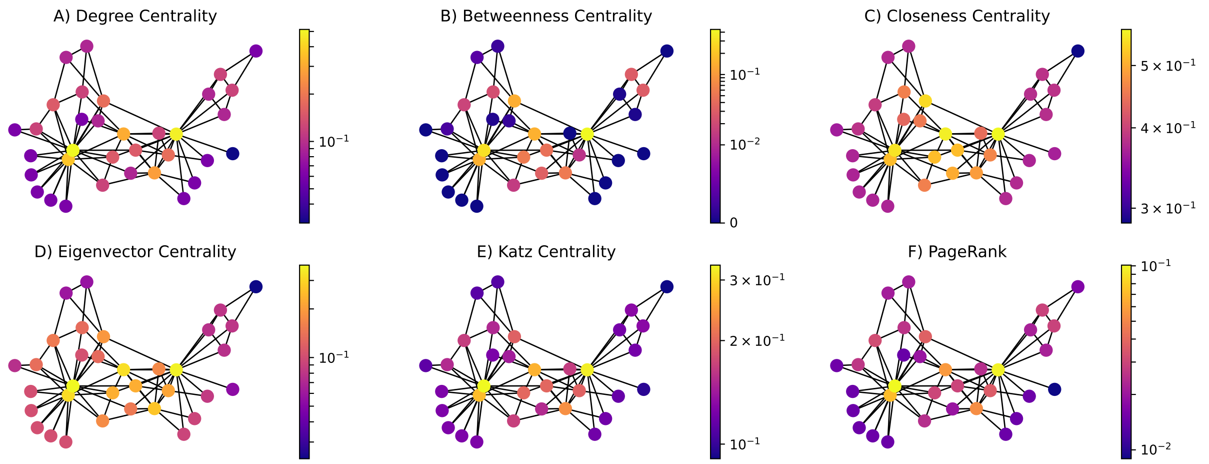


Figure 1.3: Examples of A) Degree centrality, B) Betweenness centrality, C) Closeness centrality, D) Eigenvector centrality, E) Katz centrality and F) PageRank from Zachary's Karate Club graph dataset [9].

1.3 Background on Katz centrality in static networks

Katz centrality was first introduced by Leo Katz [2] in 1953 as a way to measure the relative importance of nodes in a network based on the number of paths that pass through them. This measure is obtained by assigning a score to each node in the network based on the sum of the scores of all nodes that are one step away from it, plus a fraction of the scores of all nodes that are two steps away, and so on, up to an arbitrary limit or threshold.

Expression (1.11) provides a way to compute Katz centrality based on the *resolvent of the adjacency matrix*.

Definition 1.3.1 *Given a square matrix* $\mathbf{A}$ *its resolvent is the matrix-valued function* $R_A(\alpha) = (\mathbf{I}–\alpha\mathbf{A})^{-1}$, *defined for all* $\alpha^{-1} \in \mathbb{C} \setminus \sigma(\mathbf{A})$.

Katz centrality can be elegantly reformulated using the Neumann series as an extension of the geometric series. It can be expressed as follows

$$\mathbf{x} = (\mathbf{I} - \alpha\mathbf{A})^{-1}\mathbf{1} = \left(\sum_{k=0}^{\infty} \alpha^k \mathbf{A}^k\right)\mathbf{1}. \tag{1.12}$$

This formulation provides a practical approach for computing the resolvent of the adjacency matrix

$$(\mathbf{I} - \alpha\mathbf{A})^{-1} = \mathbf{I} + \alpha\mathbf{A} + \alpha^2\mathbf{A}^2 + \cdots + \alpha^k\mathbf{A}^k + \cdots . \tag{1.13}$$

The convergence of this expansion is ensured when $\alpha < 1/\rho(\mathbf{A})$, where $\rho(\cdot)$ represents the spectral radius.

This series is in fact the original form of centrality conceived by Leo Katz, who considered for each node i the influence of all the nodes connected by a k-length walk to i with no restriction in reuse of nodes and edges. Thus, α can be considered an attenuation parameter as the probability that an edge is successfully traversed, penalizing those nodes furthest away from i.

Considering messages being passed along the directed edges, one important consequence of the above expansion is that elements of the resolvent matrix can be considered as a measure of the ability for a node i to pass information to j taking into account all possible routes, with longer ones given less importance. In that sense, if we consider row sums in the resolvent matrix as a linear combination of powers of **A**, we can talk about the *broadcast centrality vector* (**b**) as the ability to send information for each node in the network

$$\mathbf{b} = (\mathbf{I} - \alpha\mathbf{A})^{-1}\mathbf{1}. \tag{1.14}$$

Similarly, the column sums of the resolvent matrix give a notion of the ability to receive information which is defined as the *receive centrality vector* (**r**) of the network

$$\mathbf{r} = (\mathbf{I} - \alpha\mathbf{A})^{-T}\mathbf{1}. \tag{1.15}$$

Broadly speaking, a node with a high Katz broadcast centrality will be an effective starting point for spreading a rumor, and a node with high Katz receive centrality will be an ideal location to receive the latest rumor.

In summary, Katz Centrality is a very useful centrality measure, for both directed and undirected networks, because it provides a nuanced and flexible way to assess the importance of nodes in a network based not only on their position but also on the indirect influence of a node's neighbors, which can be important in many real-world networks.

1.4 Motivation of the study

Dynamic networks, or what is the same, systems involving transient interactions are commonly found in real problems across various fields. Currently, the most popular approach is to examine network activity over discrete time frames or snapshots and analyze network status at these time slices. This method presents a number of challenges when it comes to modeling and computing, as it fails to account for the time-sensitive nature of network connections. If the time frame is too large, the ability to reproduce high-frequency transient behaviour, where an edge switches on and off multiple times in the space of a single window, is lost. On the other hand, if it is too narrow, it could result in a large number of empty time frames that can lead to redundant processes, wasting computational effort. Additionally, when time windows are too finely spaced, a static model may give a false impression of accuracy since it is not able to reflect altogether the time at which instantaneous information is sent, then received and later processed in time, as happens in many human communication media, with the subsequent loss of information in the network.

Therefore, to address these limitations the present work analyzes a continuous-time framework developed in [10] that can directly extract centrality information from a network's time-dependent adjacency matrix. This new centrality system expands the concept of the well-known Katz measure and allows us to identify and monitor the most influential nodes in dynamic networks over time at any level of detail in a natural and efficient way.

Chapter 2

Continuous-time analysis of dynamical systems

When analyzing the importance of nodes within a network, it is significant to consider the order in which node interactions occur in time. It happens that if person A meets person B today and then person B meets person C tomorrow, a message or idea could pass from A to C, but not the other way around. If we only look at individual moments in time (static snapshots) or a summary of all the interactions, we may miss this type of influence, making it difficult to identify key players in the network. It is reasonable to assume that influential nodes will introduce information that is then passed around the network by others. Our main motivation is therefore to introduce a framework that can efficiently measure this kind of dynamic effects in a continuous-time setting.

2.1 Dynamical system equations

Consider a time-ordered sequence $t_0 < t_1 < \cdots < t_M$ and its associated sequence of unweighted graphs defined over a set of N nodes, $\{G^{[k]}\}$ for $k = 0, 1, \ldots, M$. Each graph reflects the state of the network at time t_k represented by its corresponding adjacency matrix $\mathbf{A}(t_k)$, with $\mathbf{A}(t_k)_{ij} = 1$ if there is a link from i to j at time t_k, and 0 otherwise. We further assume the existence of directed links, $\mathbf{A}(t_k)_{ij} \neq \mathbf{A}(t_k)_{ji}$ but no presence of self loops, $\mathbf{A}(t_k)_{ii} \equiv 0$. We let $\Delta t_i := t_i - t_{i-1}$ denote the spacing between successive time points, not assuming that the time points are equally spaced. Then, in order to address the previously described follow-on effect derived from the time ordering, the static graph concept of walk is generalized as follows [11]:

Definition 2.1.1 *A dynamic walk of length w from node i_1 to node i_{w+1} consists of a sequence of edges $(i_1, i_2, \ldots, i_{w+1})$ and a non-decreasing sequence of times $t_{r_1} \leq t_{r_2} \leq \cdots \leq t_{r_w}$ such that $\mathbf{A}(t_{r_m})_{i_m, i_{m+1}} \neq 0$ for $m = 1, 2, \ldots, w$. The lifetime of a dynamic walk is defined by $t_{r_w} - t_{r_1}$.*

Note that more than one edge can share a time slot and that time slots must be ordered to respect the arrow of time but they do not need to be consecutive, i.e. some times may have not been used during the walk.

A key observation that generalizes the static walk mentioned in (1.13) is that the matrix product $\mathbf{A} = \mathbf{A}(t_{r_1})\mathbf{A}(t_{r_2})\cdots\mathbf{A}(t_{r_w})$ has elements $\mathbf{A}_{ij}$ that count the number of dynamic walks of length w from node i to node j on which the m'th step of the walk takes place at time t_{r_m}. In this new dynamic environment, we can utilize the same reasoning that was employed to calculate the Katz centrality metric. Our aim is to measure how likely it is for node i to engage in communication or interactions with node j. To achieve this, we can count the number of dynamic walks that go from node i to node j for each length w, reducing its significance by multiplying them with a downweighting factor α^w. This leads to the matrix product $\alpha^w\mathbf{A}(t_{r_1})\mathbf{A}(t_{r_2})\cdots\mathbf{A}(t_{r_w})$ for $t_{r_1} \leq t_{r_2} \leq \cdots \leq t_{r_w}$ which motivates the definition of the *dynamic communicability matrix*

$$\mathbf{Q}(t_M) \coloneqq (\mathbf{I} - \alpha\mathbf{A}(t_0))^{-1}(\mathbf{I} - \alpha\mathbf{A}(t_1))^{-1}\cdots(\mathbf{I} - \alpha\mathbf{A}(t_M))^{-1}, \tag{2.1}$$

or equivalently expressed by iteration,

$$\mathbf{Q}(t_k) = \mathbf{Q}(t_{k-1})(\mathbf{I} - \alpha\mathbf{A}(t_k))^{-1}, \quad k = 0, 1, \ldots, M \tag{2.2}$$

with $\mathbf{Q}(t_{-1}) = \mathbf{I}$.

To assure convergence of each resolvent matrix, as in the static network case, the parameter α is assumed to satisfy $0 < \alpha < 1/\rho^*$ where $\rho^* = \max\limits_{k=0:M}\{\rho(\mathbf{A}(t_k))\}$ is the largest spectral radius among the spectral radii of the matrices $\{\mathbf{A}(t_k)\}$. Here α plays the same role as in classical Katz centrality, i.e. the probability that a message successfully traverses an edge. In fact, the static Katz centrality is a particular case of (2.1) for $k = 0$.

The requirement of $\alpha < 1/\rho^*$ ensures that resolvents in (2.1) exist and can be expanded as $(\mathbf{I} - \alpha\mathbf{A}(t_k))^{-1} = \sum_{w=0}^{\infty}(\alpha\mathbf{A}(t_k))^w$. It follows that the entries $\mathbf{Q}(t_k)_{ij}$ represent a weighted sum of the number of dynamic walks from i to j using the ordered sequence of matrices $\{\mathbf{A}(t_0), \mathbf{A}(t_1), \ldots, \mathbf{A}(t_k)\}$ penalizing walks of length w by a factor of α^w. Hence, $\mathbf{Q}(t_k)_{ij}$ provides an overall measure of the ability of node i to send messages to node j with longer walks having less influence than shorter ones.

It is important to note that the use of $\mathbf{Q}(t_k)$ is closely linked to the concept of a starting point t_0 and an ending point t_M, that is, any walk that occurred within the time frame of t_0 to t_M holds the same level of influence in terms of time. In other words, all the factors in $\alpha^w\mathbf{A}(t_0)\mathbf{A}(t_1)\cdots\mathbf{A}(t_M)$ have the same consideration of relevance regardless of the time in which they occur. There is no time damping or temporal downweighting of any kind as previously introduced for walk lengths. This feature is appropriate for certain applications, but many other prioritize current and recent activity, disregarding activity from a distant past, as messages can become outdated, rumors lose relevance, or certain viruses become less contagious.

Similar to the concept of the walk-downweighting parameter α, Grindrod & Higham [12] considered the use of age-downweighting in the construction of the communicability matrix to further account for the decay of the relevance of the information. In their study, the researchers adopted a perspective focused on determining whether node i has recently been able to communicate with node j through short walks. They explored two contrasting extremes to capture this information. Firstly, matrix $\mathbf{A}(t_k)$ in (2.2) provides a localized snapshot, revealing the possibilities achievable through single steps based on the current day's connectivity. Conversely, matrix $\mathbf{Q}(t_{k-1})$ offers a historical outlook, encompassing all walks across all past connections leading up to the present time. This idea gives rise to a new a matrix iteration that bridges the gap between these two extremes.

In search of a time-varying "running summary" of communicability between pairs of nodes at each moment in time, our goal is to measure the ability of a node i to transfer messages to node j considering two conditions:

(i) Shorter walks are more relevant than longer walks.

(ii) Walks that commenced recently are more relevant than those that began a while ago.

These conditions motivate the concept of a *running dynamic communicability matrix*, $\mathbf{S}(t) \in \mathbb{R}^{N\times N}$ that generalize (2.2), such that $\mathbf{S}(t)_{ij}$ quantifies the ability of node i to communicate with node j up to time t [12],

$$\mathbf{S}(t_k) = \left(\mathbf{I} + e^{-\beta\Delta t_k}\mathbf{S}(t_{k-1})\right)\left(\mathbf{I} - \alpha\mathbf{A}(t_k)\right)^{-1} - \mathbf{I}, \quad k = 0, 1, 2, \ldots \tag{2.3}$$

starting for convenience with $\mathbf{S}(t_{-1}) = \mathbf{0}$, where, $\alpha \in (0, 1)$ and $\beta > 0$. Here, α is used to downweight walks of length w by the factor α^w and β is employed to reduce the weight of the activity, which is age-dependent by the factor $e^{-\beta t}$ if we consider the current age, t, of a dynamic walk as the time that has elapsed since the walk began. This factor $e^{-\beta\Delta t_k}$ in (2.3) may be interpreted as the probability that a message does not become "irrelevant" over a time length Δt_k. It is worth mentioning that by taking $\Delta t_k = 1$ for all k and $\beta = 0$ (no down-scaling in time i.e. infinitely-long memory) the communicability matrix in (2.3) recovers the original iteration product form in (2.2) with $\mathbf{S}(t_k) = \mathbf{Q}(t_k) - \mathbf{I}$. On the other hand, for $\Delta t_k = 1$ and $\beta \to \infty$, that is, $e^{-\beta\Delta t_k} \to 0$ (complete downscaling in time or zero memory), the communicability matrix yields $\mathbf{S}(t_k) = (\mathbf{I} - \alpha\mathbf{A}(t_k))^{-1}$ reducing to static Katz centrality.

So far, we have considered an environment where a fixed grid of time points are chosen but our aim, as previously stated, is to develop a new continuous-time framework. On that basis, $\mathbf{S}(t)$ is proposed to be updated over a small time interval δt, using the scaling $(\mathbf{I} - \alpha\mathbf{A}(k\delta t))^{-\delta t}$ [10]. Then, (2.3) can be rewritten as

$$\mathbf{S}(t + \delta t) = \left(\mathbf{I} + e^{-\beta\delta t}\mathbf{S}(t)\right)\left(\mathbf{I} - \alpha\mathbf{A}(t + \delta t)\right)^{-\delta t} - \mathbf{I} \tag{2.4}$$

with $\mathbf{S}(0) = \mathbf{0}$.

To ensure the meaningfulness of the $\delta t \to 0$ limit, a crucial aspect of formulating this new iteration (2.4) involved developing such an appropriate scaling approach. The key idea for employing a δt-dependent power in the matrix products becomes evident when examining the $\beta = 0$ case within a short-time interval where downscaling in time is meaningless. By refining the original single interval $[t, t + \delta t]$ into a pair of intervals $[t, t + \delta t/2]$ and $[t + \delta t/2, t +$

$\delta t]$, and assuming $\mathbf{A}(t)$ remains constant during this interval, we can observe the following straightforward relationship

$$(\mathbf{I}-\alpha\mathbf{A}(t))^{-\frac{\delta t}{2}}(\mathbf{I}-\alpha\mathbf{A}(t))^{-\frac{\delta t}{2}}=(\mathbf{I}-\alpha\mathbf{A}(t))^{-\delta t}.$$

This identity serves to demonstrate the consistency of our chosen scaling approach, thereby producing reliable and coherent results with (2.1).

Now, for practical purposes we define the *principal logarithm* of a matrix [13, Ch. 11].

Definition 2.1.2 *A logarithm of* $\mathbf{A}\in\mathbb{C}^{N\times N}$ *is any matrix* $\mathbf{X}\in\mathbb{C}^{N\times N}$ *such that* $e^{\mathbf{X}}=\mathbf{A}$ *where* $e^{\mathbf{X}}=\mathbf{I}+\mathbf{X}+\frac{\mathbf{X}^2}{2!}+\frac{\mathbf{X}^3}{3!}+\cdots$. *The matrix logarithm is not unique, since* $e^{\mathbf{X}+2k\pi i\mathbf{I}}=\mathbf{A}$ *for any integer* k. *If* $\mathbf{A}$ *has no negative real eigenvalues, then we call this the principal logarithm of* $\mathbf{A}$*, which is the unique logarithm whose spectrum lies in the strip* $\{z:-\pi<Im(z)<\pi\}$.

From this point on, we will assume the principal logarithm when referring to the logarithm of a matrix.

For convenience, we take the identity matrix to the left hand side from (2.4), and define the *communicability matrix* $\mathbf{U}(t)=\mathbf{I}+\mathbf{S}(t)$. Applying the matrix exponential and matrix logarithm with the identities [13, Ch. 11]

$$\mathbf{H}=e^{\log\mathbf{H}}\ \text{ and }\ \log\mathbf{H}^{\alpha}=\alpha\log\mathbf{H}\ \text{ for }\ -1\le\alpha\le 1,$$

we obtain the following identity from (2.4):

$$\mathbf{U}(t+\delta t)=\left(\mathbf{I}+e^{-\beta\delta t}(\mathbf{U}(t)-\mathbf{I})\right)\exp\left(-\delta t\log(\mathbf{I}-\alpha\mathbf{A}(t+\delta t))\right). \tag{2.5}$$

Expanding in a Taylor series the right-hand side of (2.5) to the first order

$$\begin{aligned}
f(\delta t)&=\left(\mathbf{I}+e^{-\beta\delta t}(\mathbf{U}(t)-\mathbf{I})\right)\exp\left(-\delta t\log(\mathbf{I}-\alpha\mathbf{A}(t+\delta t))\right)\\
f(0)&=\mathbf{U}(t)\\
f'(\delta t)&=-\beta(\mathbf{U}(t)-\mathbf{I})e^{-\beta\delta t}\exp\left(-\delta t\log(\mathbf{I}-\alpha\mathbf{A}(t+\delta t))\right)+\\
&\quad+(\mathbf{I}+e^{-\beta\delta t}(\mathbf{U}(t)-\mathbf{I}))(-\log(\mathbf{I}-\alpha\mathbf{A}(t+\delta t)))\exp(-\delta t\log(\mathbf{I}-\alpha\mathbf{A}(t+\delta t)))\\
f'(0)&=-\beta(\mathbf{U}(t)-\mathbf{I})-\mathbf{U}(t)\log(\mathbf{I}-\alpha\mathbf{A}(t)),
\end{aligned}$$

and rearranging the terms, (2.5) can be rewritten as

$$\begin{aligned}
\mathbf{U}(t+\delta t)&=\mathbf{U}(t)+\left(-\beta(\mathbf{U}(t)-\mathbf{I})-\mathbf{U}(t)\log(\mathbf{I}-\alpha\mathbf{A}(t))\right)\delta t+\mathcal{O}(\delta t^2)\\
\frac{\mathbf{U}(t+\delta t)-\mathbf{U}(t)}{\delta t}&=-\beta(\mathbf{U}(t)-\mathbf{I})-\mathbf{U}(t)\log(\mathbf{I}-\alpha\mathbf{A}(t))+\mathcal{O}(\delta t)
\end{aligned}$$

By taking the limit $\delta t \to 0$, we finally arrive at the matrix ODE proposed by Grindrod and Higham [10],

$$\begin{cases} \mathbf{U}'(t) = -\beta(\mathbf{U}(t) - \mathbf{I}) - \mathbf{U}(t)\log(\mathbf{I} - \alpha\mathbf{A}(t)), \quad t > 0 \\ \mathbf{U}(0) = \mathbf{I}. \end{cases} \tag{2.6}$$

This matrix ODE (2.6) provides us with a continuous-time framework for a dynamic system driven by its adjacency matrix, $\mathbf{A}(t)$ with $\mathbf{U}(t)_{ij}$ for $i \neq j$ quantifying the current ability of node i to pass information to node j, so that longer and older walks are less important.

In the same way as we did with Katz centrality for static networks in (1.14–1.15), we define two dynamic vectors, namely the broadcast vector $\mathbf{b}(t)$ and the receive vector $\mathbf{r}(t)$,

$$\mathbf{b}(t) = \mathbf{U}(t)\mathbf{1} \quad \text{and} \quad \mathbf{r}(t) = \mathbf{U}(t)^T\mathbf{1}. \tag{2.7}$$

These vectors enable us to measure, respectively, the current inclination of each node to broadcast or receive information across a dynamic network, under the assumptions of less significance to longer and older walks.

2.2 Remarks on the new framework

Real-time updating of the receive centrality is approximately a factor N simpler than real-time updating of broadcast centrality for sparse networks.

Simulating the dynamic broadcast centrality vector, $\mathbf{b}(t)$, is computationally much more expensive than the dynamic receive vector. In (2.6), $\mathbf{b}(t)$ requires to work with the full matrix $\mathbf{U}(t)$ which implies to deal with orders of $\mathcal{O}(N^2)$ for storage and an $\mathcal{O}(N^2)$ cost per unit time. One possible approach to address this problem is to develop approximation techniques, such as sparsifying $\mathbf{U}(t)$ or reducing its dimension.

On the other hand, we note that the receive centrality vector, $\mathbf{r}(t)$, satisfies its own vector-valued ODE. If we transpose both sides of the equality in (2.6) and multiply on the right by the vector of ones, we obtain

$$\mathbf{r}'(t) = -\beta(\mathbf{r}(t) - \mathbf{1}) - (\log(\mathbf{I} - \alpha\mathbf{A}(t)))^T\mathbf{r}(t) \tag{2.8}$$

with $\mathbf{r}(0) = \mathbf{1}$. This implies a considerable reduction of order $\mathcal{O}(N)$ with respect to equation (2.6) if we assume that $\mathbf{A}$ represents the adjacency matrix for a sparse network with a computational cost that grows linearly with the number of nonzero entries.

Unfortunately, it is not possible to derive a vector-valued ODE for the dynamic broadcast vector using the same method. This dissimilarity comes from the fact that the receive vector

$\mathbf{r}(t)$ monitors the total amount of information that flows into each node, so this information can be carried forward in time as new links emerge. In contrast, the broadcast vector $\mathbf{b}(t)$ tracks the information that has left each node but it does not indicate the current location of the information because this has not been recorded, and therefore, we cannot update it based solely on $\mathbf{b}(t)$.

The total dynamic broadcast centrality and the aggregate network centrality can be computed via the dynamic receive vector $\mathbf{r}(t)$ *for large, sparsely connected networks at a reasonable computational cost.*

The *total broadcast centrality* of the network, which is represented by $\sum_{i=1}^{N} \mathbf{b}(t)_i$, and the *aggregate network centrality*, $\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{U}(t)_{ij}$ can be computed using equation (2.8). This is because in any matrix, the sum of row sums is equal to the sum of column sums. In our case $\sum_{i=1}^{N} \mathbf{b}(t)_i = \sum_{i=1}^{N} \mathbf{r}(t)_i = \sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{U}(t)_{ij}$. Therefore, by running a simulation of an ODE system that is N times smaller than the one needed for nodal broadcast information, we can keep track of the current broadcast capability of the entire network by computing $\mathbf{r}(t)^T\mathbf{1}$. The storage requirement for $\mathbf{r}(t)$ scales like the number of nodes in the system, $\mathcal{O}(N)$, and the primary computation involved in evaluating the right-hand side of (2.8) can be accomplished by performing only a few products of a sparse matrix with a full vector. This results in a cost of $\mathcal{O}(N)$ per unit time for a network with $\mathcal{O}(1)$ edges per node.

While total broadcast centrality or aggregate centrality are useful measures for gaining different perspectives on a network at a general level, this study will focus specifically on determining centrality at a node level.

The choice of downweighting parameters, α *and* β*, plays an important role in (2.6) and it is strongly connected to the nature of the interactions represented by* $\mathbf{A}(t)$.

On one side, the value of the β parameter, which downweights temporal information, can be interpreted as the rate at which information becomes less relevant over time. In our model, this means that walks starting t time units ago are downweighted by $e^{-\beta t}$. The question that now arises is how we can calibrate this β parameter.

A recent study [14] by Bit.ly, the URL shortening service, reveals that the average lifespan of a link online is surprisingly short. The research found that most links, regardless of genre, experience a brief burst of attention followed by a rapid decline in engagement. The study measured a link's longevity by calculating its half life, i.e., the point at which it receives half of its total clicks.

According to the study, links shared on various platforms have different lifespans. On Twitter, a link reaches its half life in an average of 2.8 hours, while on Facebook it lasts slightly longer at 3.2 hours. Links shared through email and messenger services have the longest lifespan of 3.4 hours. Notably, YouTube links have a significantly longer half life of 7.4 hours, indicating sustained interest from users.

Therefore, by examining the typical half-life of network links, it becomes feasible to determine an appropriate value for β, representing the rate at which information becomes outdated.

On the other hand, the value of the edge-attenuation parameter α, which penalizes longer walks in the network, is determined in the discrete case as we saw in (2.1) by the reciprocal of the largest spectral radius among the matrices $\{\mathbf{A}(t_k)\}$. Similarly, for the continuous-time ODE system (2.6), the principal matrix logarithm function $\log(\mathbf{I} - \alpha\mathbf{A}(t))$ is well-defined when [13, Ch. 11]

$$1 - \alpha\rho(\mathbf{A}(t)) > 0 \iff \alpha < \rho(\mathbf{A}(t))^{-1} \quad \text{for all} \quad t > 0.$$

This implies that, much like in the discrete case, the full temporal evolution of the network represented by $\mathbf{A}(t)$ has to be known for every t before deriving $\mathbf{U}(t)$.

As a particular example of the choice of α, let us consider the context of undirected one-to-one communication, such as voice calls with no teleconferences, i.e., no more than one active connection per node at a given time. It is observed that the the adjacency matrix $\mathbf{A}(t)$ for this kind of network can always be permuted into a block diagonal structure, with non-trivial blocks of the form

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

where $\lambda_1 = 1$.

Consequently, the constraint on α is reduced to $\alpha < 1$. This condition will be applied for the choice of the α parameter when dealing with the second and third numerical experiments of this thesis, both of which simulate this type of communication network.

Alternative approaches to resolvent based centrality measures arise by replacing the resolvent function with an opportune matrix function.

The initial assumption in (1.13) relies on the concept of counting dynamic walks in a broad sense, encompassing any path that utilizes zero, one, or multiple edges per time step. Alternatively, we could adopt a different approach in which we count other types of dynamic walks or matrix functions. One possible way to begin this alternative approach is by using the following iteration technique

$$\mathbf{S}(t+\delta t) = (\mathbf{I} + e^{-\beta\delta t}\mathbf{S}(t))(H(\mathbf{A}(t))^{-\delta t} - \mathbf{I} \tag{2.9}$$

where $H(\mathbf{A}(t))$ is some matrix function. One suitable option for this matrix function is to use truncated power series such as $\mathbf{I}+\alpha\mathbf{A}(t)+\alpha^2\mathbf{A}(t)^2+\cdots+\alpha^p\mathbf{A}(t)^p$, which only consider dynamic walks that involve a maximum of p edges per time step. When employing this approach, the ODE system (2.6) takes on a more general form

$$\mathbf{U}'(t) = -\beta(\mathbf{U}(t) - \mathbf{I}) + \mathbf{U}(t)\log(H(\mathbf{A}(t))) \tag{2.10}$$

Truncated power series for the resolvent matrix of the adjacency matrix $\mathbf{A}(t)$ will be used later in this work to approximate the computation of the matrix logarithm in networks involving a large number of nodes.

Chapter 3

Numerical experiments

From the point of view of computational cost, it is convenient to make a brief analysis of the relevant new continuous-time framework obtained in (2.6). The main problem when dealing with large networks (matrices) resides in the computation of the matrix logarithm, which can be computationally expensive.

The matrix logarithm can be approximated by its Taylor series expansion. The Taylor series expansion of the matrix logarithm of $\mathbf{I} - \alpha\mathbf{A}$ can be expressed as:

$$\log(\mathbf{I} - \alpha\mathbf{A}) \approx \sum_{k=1}^{n} (-1)^{k+1} \frac{\alpha^k}{k} \mathbf{A}^k$$

where $\mathbf{I}$ is the identity matrix of size N. The accuracy of the approximation depends on the value of k, which determines how many terms in the series are included. The higher the value of k, the more accurate the approximation, but also the more computationally expensive it is to compute.

It is worth noting that for certain matrices, the Taylor series expansion does not converge or converges too slowly to be practical for approximating the matrix logarithm. In such cases, other approximation methods like the Padé approximation or Schur decomposition may be more effective [13, Ch. 11].

In Python, the `scipy.linalg.logm` module from the SciPy library computes the logarithm of a matrix using a inverse scaling and squaring method together with a Schur decomposition with a computational cost of $\mathcal{O}(N^3)$ for an $N \times N$ matrix. However, the Schur decomposition is not always the most efficient way to compute the matrix logarithm, especially for large matrices with special properties such as sparsity or symmetry. In these cases, specialized algorithms, based on Krylov methods, rational approximations or iterative methods with preconditioning, may be used to approximate the matrix logarithm to reduce its computational cost.

In this section, we consider first the two synthetic experiments from [10] in order to demonstrate how our new matrix ODE approach works and provide a better understanding of the α, β parameters. The *synthetic* concept in these two experiments refers to the fact that they are based on artificially created networks with a simple and well-known structure of nodes. By comparing expected results with the actual ones obtained in the experiments, we will be able to test, validate and refine the new dynamic framework. The relatively small size of these examples, $N = 31$ and $N = 17$ nodes respectively, will facilitate the process of visualizing, and it will also allow us to employ a highly precise Runge-Kutta iteration to solve the corresponding ODE systems (2.6) with accuracy.

The new model is also examined in a third experiment based on real voice call data from the IEEE VAST 2008 Challenge [15]. Here, we try to find out if the new dynamic measures of centrality are able to identify hubs of influence in a large communication network ($N = 400$), and if they offer a better perspective of the network evolution than static or aggregate measures.

For the initial two experiments, where there are only a few nodes, the Schur decomposition-based `scipy.linalg.logm` Python function is used to compute the matrix logarithm. However, when dealing with a larger number of nodes, as in the voice call experiment, a Taylor series approximation is considered to be a more effective approach to enhance computational efficiency.

The Python code used for the three experiments is provided in the appendix A of this thesis. Each experiment involves the definition and analysis of the adjacency matrices generated for the different time intervals. Subsequently, the α and β parameters are established to implement the matrix ODE function (2.6). Such a system of differential equations is then solved using the Scipy's function `solve_ivp`, employing Runge-Kutta methods. Furthermore, the thesis includes a comparative analysis of the computational costs associated with the computation of $\mathbf{b}(t)$ and $\mathbf{r}(t)$.

3.1 First synthetic experiment

The first synthetic experiment models a cascade of information through the directed binary tree structure with $N = 31$ nodes illustrated in Fig. 3.1. On a time interval $t \in [0, 20]$, the adjacency matrix $\mathbf{A}(t)$ of such network switches ten times between two constant values $\mathbf{A}_{even}$ and $\mathbf{A}_{odd}$ on each sub-interval $[i, i+1)$ for $i = 0, 1, 2, \dots$, specifically

$$\mathbf{A}(t) := \begin{cases} \mathbf{A}_{even}, & \text{if} \quad \text{mod } (\lfloor t \rfloor, 2) = 0 \\ \mathbf{A}_{odd}, & \text{otherwise} \end{cases}$$

where $\lfloor t \rfloor$ denotes the floor function, $\mathbf{A}_{even}$ the adjacency matrix relative to the subgraph with solid edges in Fig. 3.1, and $\mathbf{A}_{odd}$ the one relative to the subgraph with dashed edges. During the time an edge from node i to node j is active the respective entry of the adjacency matrix is set to $\mathbf{A}(t)_{ij} = 1$, and zero otherwise. As we are dealing with directed links, $\mathbf{A}(t)_{ij} \neq \mathbf{A}(t)_{ji}$. This results in asymmetric matrices $\mathbf{A}(t) \in \mathbb{R}^{31\times 31}$ where most of their elements are zero and only a few entries belonging to the active nodes are set to one. More specifically, the non-zero entries of each adjacency matrix are given by $\mathbf{A}_{i,2i} = \mathbf{A}_{i,2i+1} = 1$ for $i \in \mathbb{S}(t)$, where

$$\mathbb{S}(t) = \begin{cases} \{1,4,5,6,7\} & \text{if} \quad \text{mod}\,(\lfloor t \rfloor, 2) = 0, \\ \{2,3,8,9,10,11,12,13,14,15\} & \text{otherwise} \end{cases}$$

Some noise is added to this structure by including extra directed edges that are chosen uniformly at random for each subinterval, with an average of five edges added each time (see Appendix A - Python code, l.28).

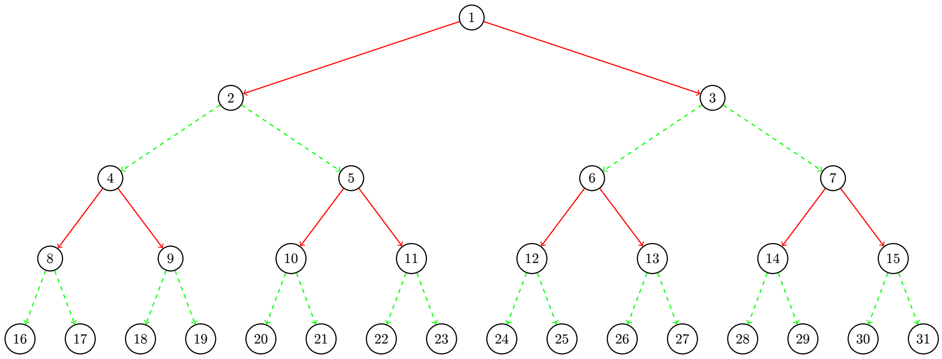


Figure 3.1: Network structure (binary tree) for the first synthetic experiment. The active links of $\mathbf{A}(t)$ alternate between the solid and dashed edges, with extra noise added at each time step, over a period of 10 cycles.

With this experiment, two main objectives are sought. First, to have a better understanding of what role α and β parameters play. And second, to confirm that our model captures the cascade effect in the network along a hierarchy of influence that is hidden from a static or aggregate view.

For that purpose, four experiments were run to compute the dynamic broadcast centrality at $t = 20$ for each node through (2.6)–(2.7), varying only the values of α and β as shown in Table 3.1. Note that the last experimental run does not consider any parameter as it is based on the computation of aggregate measures, in this case, the aggregate out degree represented by row sums of the aggregate adjacency matrix for the whole interval $t \in [0, 20]$. By analyzing the eigenvalues of all generated $\mathbf{A}_{even}$ and $\mathbf{A}_{odd}$ it is seen that the maximum of the spectral radii of these matrices is 1, and therefore the solution to (2.6) is well defined for $\alpha \in (0, 1)$. The SciPy's `solve_ivp` method was used to numerically solve the matrix ODE, which uses an explicit Runge-Kutta method of order 5(4). Relative and absolute tolerances were left to their default values, $atol = 10^{-6}$ and $rtol = 10^{-3}$.

Parameter	**#1**	**#2**	**#3**	**#4**
α	0.7	0.7	0.1	-
β	0.1	0.01	0.1	-

Table 3.1: First synthetic experiment: Choice of the α, β parameters for the different experimental runs.

Fig. 3.2a displays the dynamic broadcast centrality from (2.7), at time $t = 20$ for each of the 31 nodes for $\alpha = 0.7$ and $\beta = 0.1$. We observe that node 1 has a strong advantage in terms of centrality, and that centrality tends to decrease as the index increases. Nodes 2 and 5 are ranked higher than node 3, indicating that the additional noise has affected this part of the network.

Fig. 3.2b depicts the same results with a lower $\beta = 0.01$ value, which increases the contribution of older walks. As a reminder, if we consider a walk starting at time zero, then the downweighting factor becomes $e^{-20\times 0.01} \approx 0.8$ rather than $e^{-20\times 0.1} \approx 0.1$. This change of the β parameter has a negligible effect on the node rankings which makes sense considering that the network dynamics have an underlying periodic pattern, but it can be observed that it generates larger absolute values.

In Fig. 3.2c, we return to the original $\beta = 0.1$ value and set $\alpha = 0.1$, resulting in less marked differences in the node rankings. Even though node 1 continues to be the most central, the differences are less pronounced as its capability to initiate numerous dynamic walks of length 4 to nodes at the bottom of the hierarchy has less influence or weight.

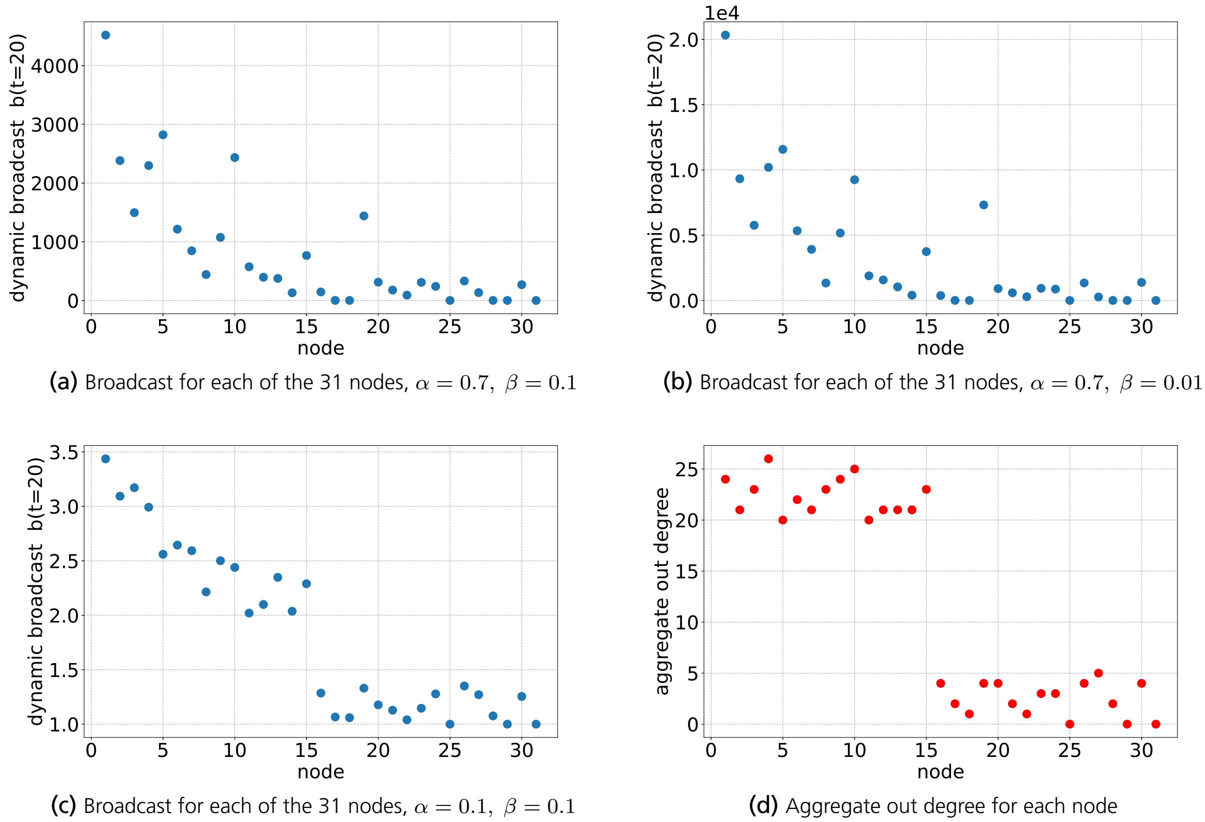
Fig. 3.2d illustrates the aggregate out degree of each node, that is, the sum of out degrees over time for each node. For nodes 1 to 15 in the binary tree structure, this value is 20 (2 active edges $\times$ 10 cycles), which fluctuates due to the introduced noise.

Overall, this experiment allows us to clearly visualize the dynamic effect arising from the time-ordering of the node interactions. Due to the hierarchy and timing of the edges, information appears to flow from lower indices to higher. The binary structure of the tree makes node 1 particularly efficient at transmitting information through the network, although this is not immediately apparent in a single snapshot. The last plot highlights that the overall bandwidth in terms of the aggregate out degree of a node can be a misleading network metric for determining node influence.

3.2 Second synthetic experiment

We now consider the second synthetic experiment from [10]. The experiment simulates multiple rounds of voice calls that occur along an undirected binary tree structure. Each node in the tree has at most one active edge at any given time, meaning that there are no "conference" calls. Fig. 3.3 shows the network of $N = 17$ nodes with labels assigned to the edges indicating when they are active. The adjacency matrix, $\mathbf{A}(t)$, for this experiment is defined based on the ordered and non-overlapping time intervals such that $t_i := [(i-1)\tau, (i-1 + 0.9)\tau)$, for $i = 0, 1, \ldots, 7$, and $\tau = 0.1$.

The dynamic network is constructed in a way that node A (node 1) is designed to be a more effective influencer compared to node B (node 2). This can be attributed to several factors such as a higher social/business status or access to more current and relevant information. Connections are built in such a way that node A talks to node C (node 3) in t_1, initiating a cascade of phone calls in the network. On the other hand, node B communicates with node D (node 17) at t_1



(a) Broadcast for each of the 31 nodes, $\alpha = 0.7,\ \beta = 0.1$

(b) Broadcast for each of the 31 nodes, $\alpha = 0.7,\ \beta = 0.01$

(c) Broadcast for each of the 31 nodes, $\alpha = 0.1,\ \beta = 0.1$

(d) Aggregate out degree for each node

Figure 3.2: First synthetic experiment: results from the dynamic network in Fig. 3.1.

and waits until t_4 to contact node C, which does not trigger any new cascades. The experiment is repeated over five cycles, during which nodes A and B send out a total of 10 messages.

What we expect from this experiment, and makes it different from the previous one, is to verify that our continuous-time framework is capable of revealing the differences in terms of dynamic broadcast centrality between the node A that enjoys a cascade effect of information in the network and node B that does not. Even if these nodes have an apparent identical behaviour from an overall perspective, both contacting nodes C and D for the same length of time.

As pointed out in the remark on the role of the α and β parameters, in the scenario of undirected one-to-one communication, such as voice calls with no teleconferences, all the symmetric adjacency matrices turn out to have unitary spectral radius for each time interval. A complete definition of these matrices can be seen in Appendix A - Second synthetic experiment, 1.22. Values of $\alpha = 0.7$ and $\alpha = 0.9$ were chosen to compare centrality results keeping fixed the value of $\beta = 0.1$ which does not play an important role in this experiment due to the underlying periodic pattern in the network dynamics (similar results were obtained for different values of β).

Implementing (2.6)–(2.7) in Python, $\mathbf{b}(t)$ was computed for the interval $t \in [0, 3.5]$ for nodes A and B, using the `solve_ivp` function with default parameters: $method$ = "$RK45$", $atol = 10^{-6}$ and $rtol = 10^{-3}$.

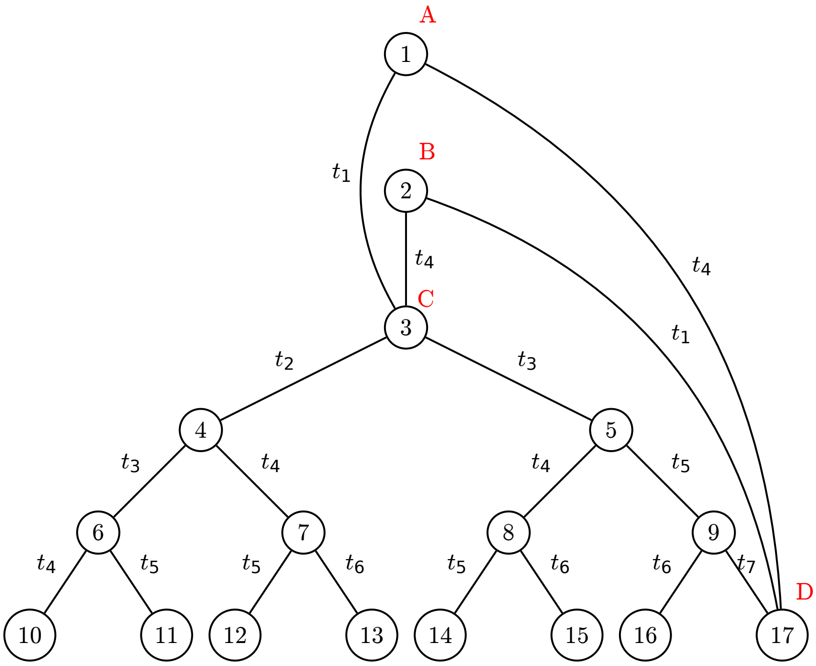
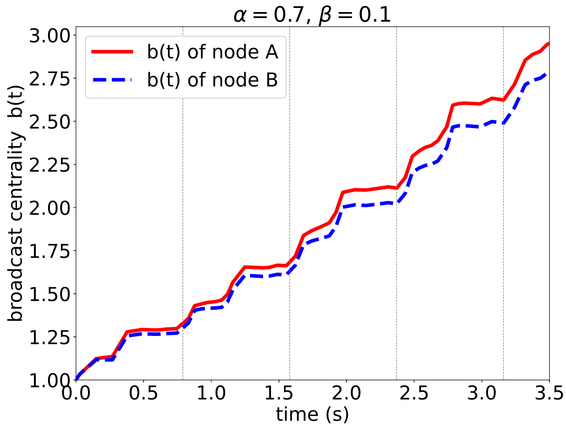


Figure 3.3: Network structure for the second synthetic experiment. Links of $\mathbf{A}(t)$ are active over non-overlapping time intervals such that $t_i := [(i\text{-}1)\tau, (i\text{-}1 + 0.9)\tau)$, for $i = 0, 1, \ldots, 7$, and $\tau = 0.1$, repeated periodically over five cycles.

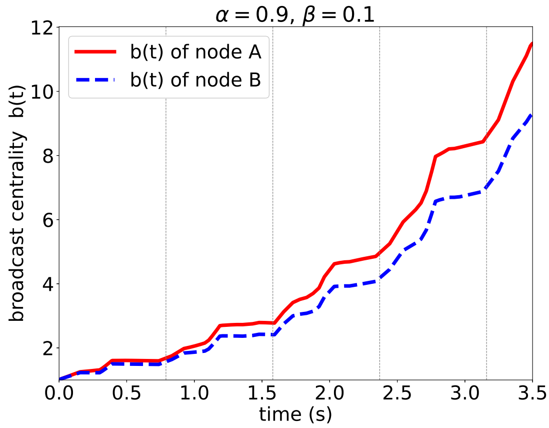
The results in Fig. 3.4 (for $\alpha = 0.7, \beta = 0.1$ and $\alpha = 0.9, \beta = 0.1$ respectively) show that the dynamic broadcast centrality measure is able to capture the cascade effect enjoyed by node A (solid line) with respect to node B (dashed line). This difference in broadcast centrality is much more pronounced for values of α close to one (see Fig. 3.4b), where longer walks are more strongly penalized.

The iterative way in which the dynamic communicability matrix (2.2) is defined as a product of non-negative matrices based on the computation of the previous steps, ensures that nodes do not become less communicative over time. This characteristic is reflected in the increasing curves for both plots. This can also be interpreted from the fact that as time goes on, more nodes in the network will have received the message. Then, as more nodes receive the message, the number of nodes that can be reached in a single step will increase, and so will the broadcast centrality of the original node. Moreover, if we divide the plots into the five cycles that network communication is repeated by vertical lines, the particular staircase shape of the curves is explained by the fact that nodes A and B are active during t_1 and t_4 which causes a notable increase in broadcast centrality after these points.

As a curiosity, other nodes in this network, such as nodes 3, 4 and 5, offer greater centrality than nodes 1 (node A) and node 2 (node B). In fact, node 3 is the one with the highest centrality in the entire network. However, the main objective of this experiment was not to highlight this feature but to verify that our continuous-time ODE framework was capable of establishing a remarkable difference in the dynamic behavior between nodes A and B.



(a) $\mathbf{b}(t)$ for $\alpha = 0.7,\ \beta = 0.1$



(b) $\mathbf{b}(t)$ for $\alpha = 0.9,\ \beta = 0.1$

Figure 3.4: Second synthetic experiment: dynamic broadcast centrality over time for node A (solid) and node B (dashed) in the network of Fig. 3.3.

3.3 Voice call experiment

The next experiment applies the new modelling framework (2.6) to a set of voice call interactions, in order to analyze the dynamic behavior of a fictitious, controversial socio-political movement. The data used for the experiment, supplied as part of the IEEE VAST 2008 Challenge [15], consists of a complete set of 9834 time-stamped calls across 400 cell phone users over a 10 day period, with information on IDs for the send and receive nodes, start time in hours/minutes, and duration in seconds.

Additional information is provided by the designers of the above mentioned competition indicating that node 200 is the leader of an important community who controls a closely connected subnetwork or inner circle consisting of nodes 1, 2, 3, and 5. However, starting from day 7, these individuals seem to switch their phone IDs: node 200 becomes 300, and the others become 306, 309, 360, and 392.

The aim of the experiment is to show the usefulness of the new matrix ODE in dealing with this type of dynamic network, comparing dynamical measures against aggregated. To that end, the bandwidth of a node is defined as the aggregate number of seconds for which the node ID is active as a sender or receiver. This will allow us to compare the effective activation time of a certain node with its relevance in terms of dynamic broadcast centrality.

For this experiment, $\mathbf{A}(t)$ is assumed to be symmetric, meaning that $\mathbf{A}(t)_{ij} = \mathbf{A}(t)_{ji} = 1$ if nodes i and j are communicating at time t, which is measured in seconds. The β parameter is chosen to be approximately $\beta = 1/(60 \times 60 \times 24) \approx 1.2 \times 10^{-5}$, which corresponds to a time downweighting of e^{-1} per day. We set the edge attenuation parameter α to a similar value of 10^{-4}. The SciPy's `solve_ivp` method is again used to numerically solve the ODE (2.6) (see Appendix A - Voice call experiment, l.257). Additionally, absolute and relative error tolerances are both set to 10^{-4}. To improve efficiency, the matrix logarithm is approximated in this occasion with its expansion to the fifth power:

$$\log(\mathbf{I} - \alpha \mathbf{A}(t)) \approx \alpha \mathbf{A}(t) - \alpha^2 \mathbf{A}(t)^2/2 + \alpha^3 \mathbf{A}(t)^3/3 - \alpha^4 \mathbf{A}(t)^4/4 + \alpha^5 \mathbf{A}(t)^5/5$$

Visually speaking, the effects of increasing the number of terms in the expansion to 6 and 7 remained identical.

This experiment yields two key findings:

(i) The dynamic broadcast/receive measures (2.7) are able to identify key nodes as highly influential, even if they are not actively using much bandwidth, without any prior knowledge of an inner circle's existence.

(ii) The transformation that occurs in the network on day 7 is revealed by the running centrality measures once we know the IDs of the inner circle.

Fig. 3.5 is used to demonstrate point (i) by plotting bandwidth against dynamic broadcast, $\mathbf{b}(t)$, using data up until the end of day 6. The scatter plot also includes symbols to mark certain nodes, such as a star for the ring leader, 200, and squares for the related inner-circle nodes, 1, 2, 3, and 5. The follow-on ID for the ringleader, 300, is marked with a plus symbol, and those for other members, 306, 309, 360, and 392, are marked with diamonds.

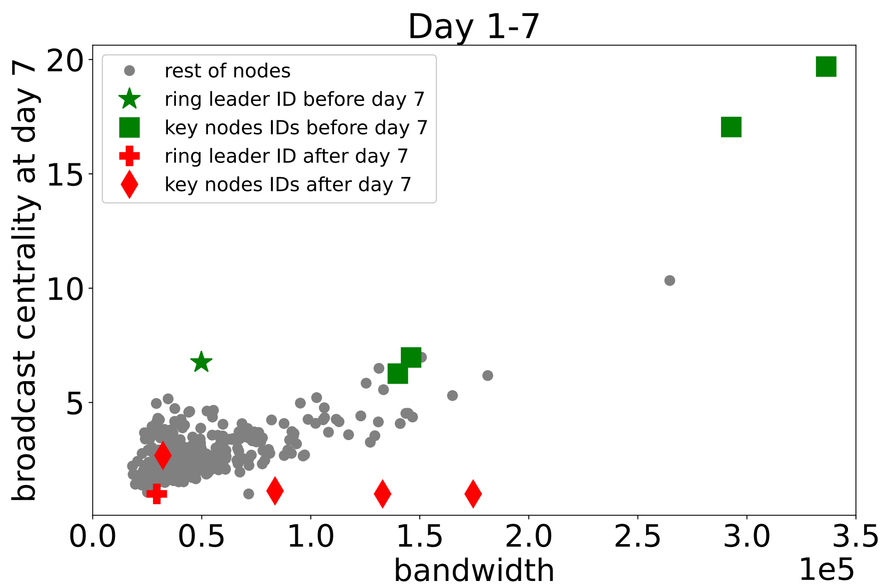


Figure 3.5: Voice call experiment: broadcast centrality for each node at the end of day 6 vs. bandwidth (secs.).

The results show that the key nodes for days 1-7 are much more dominant in terms of dynamic broadcast than overall bandwidth. Specifically, the ringleader node has a low bandwidth but ranks sixth out of 400 in terms of broadcast communicability.

The data from days 7 to 10 is displayed in Fig. 3.6. The plot shows how the new ID of the ringleader, that is indicated by a plus symbol, has a low overall bandwidth, but ranks seventh highest for broadcast centrality. Although the former IDs from the inner circle, marked with diamonds, still possess high bandwidth, their low dynamic broadcast scores suggest that they are no longer central players. This experiment confirms that the dynamic broadcast score is a more effective indicator of centrality than overall bandwidth in uncovering the inner circle.

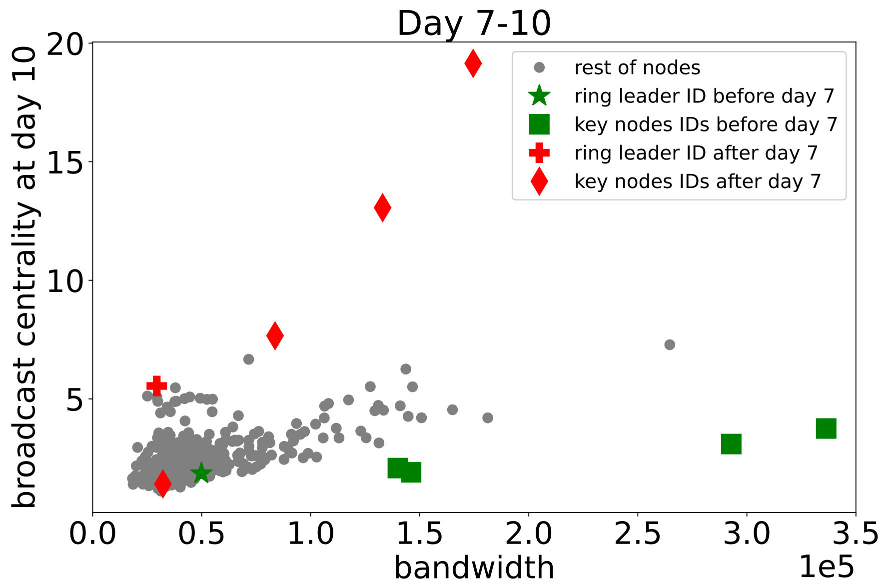


Figure 3.6: Voice call experiment: broadcast centrality for each node at the end of day 10 vs. bandwidth (secs.).

Similar results are found when the dynamic receive centrality, $\mathbf{r}(t)$, is computed by using its own vector-valued ODE (2.8). For data from days 1 to 7, a computation time reduction per function call of approximately 30% is obtained with respect to $\mathbf{b}(t)$ computations (see Appendix A - $\mathbf{b}(t)$ vs. $\mathbf{r}(t)$ cost comparison). The execution times in that comparison are measured over 100 repetitions for the computation of their respective matrix/vector ODE systems. The frequency (time per call ratio) at which the matrix function (2.6) or vector function (2.8) is evaluated by the `solve_ivp` Python method with parameters $RK45, rtol = 0.05, atol = 0.01$, and $RK45, rtol = atol = 10^{-4}$ is shown in Table 3.2 and Table 3.3 respectively.

Method (RK45 $\times$ 100 rep.)	rtol/atol	time (s)	function calls	ratio (ms/call)
$\mathbf{b}(t)$	0.05/0.01	241.24	9200	26.22
$\mathbf{r}(t)$	0.05/0.01	204.35	11600	17.62

Table 3.2: $\mathbf{b}(t)$ vs. $\mathbf{r}(t)$ cost comparison for $RK45, rtol = 0.05, atol = 0.01$.

For smaller error tolerance values (Table 3.3), the results reveal that the vector ODE for $\mathbf{r}(t)$ is invoked significantly more times than the matrix ODE needed for $\mathbf{b}(t)$. The underlying reasons for this observation would demand a comprehensive investigation of the internal workings of the Python `solve_ivp` algorithm, which is beyond the scope of this thesis. Just mention here that for the tolerance scale proposed in Table 3.2, where the number of evaluations is similar, the improvements mentioned in this thesis for $\mathbf{r}(t)$ are remarkable both in terms of absolute time values and the time per call ratio.

Method (RK45 $\times$ 100 rep.)	**rtol/atol**	**time (s)**	**function calls**	**ratio (ms/call)**
$\mathbf{b}(t)$	$10^{-4}/10^{-4}$	984.47	32600	30.20
$\mathbf{r}(t)$	$10^{-4}/10^{-4}$	2627.48	122600	21.43

Table 3.3: $\mathbf{b}(t)$ vs. $\mathbf{r}(t)$ cost comparison for $RK45, rtol = atol = 10^{-4}$.

In order to support point (ii), we define the *communicability between key nodes* ($\mathcal{C}$) at each discretized time point, as the average amount of messages, broadcast ($\mathbf{U}_{ij}$) and received ($\mathbf{U}_{ji}$), between each pair of key nodes, scaled by the average amount of communication between all pairs of nodes in the network, i.e.,

$$\mathcal{C}(t) = \frac{\overline{K}(t)}{\overline{T}(t)},$$

where

$$\overline{K}(t) = \frac{\sum_{i\neq j} \left(\mathbf{U}(t)_{ij} + \mathbf{U}(t)_{ji}\right)}{2\sum_{k=1}^{4} k} \quad \text{for } \; i = 1, 2, 3, 5, 200,$$
$$\overline{T}(t) = \frac{\left(\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{U}(t)_{ij}\right) - \sum_{i=1}^{N} \mathbf{U}(t)_{ii}}{N^2 - N}.$$

The communicability between the original IDs of the five most important players (ID nodes: 200, 1, 2, 3, and 5) during the ten days period is presented in Figure 3.7. By analyzing this measure, we can observe how the structure of the network changes over time, especially when the players start using different IDs after day 7.

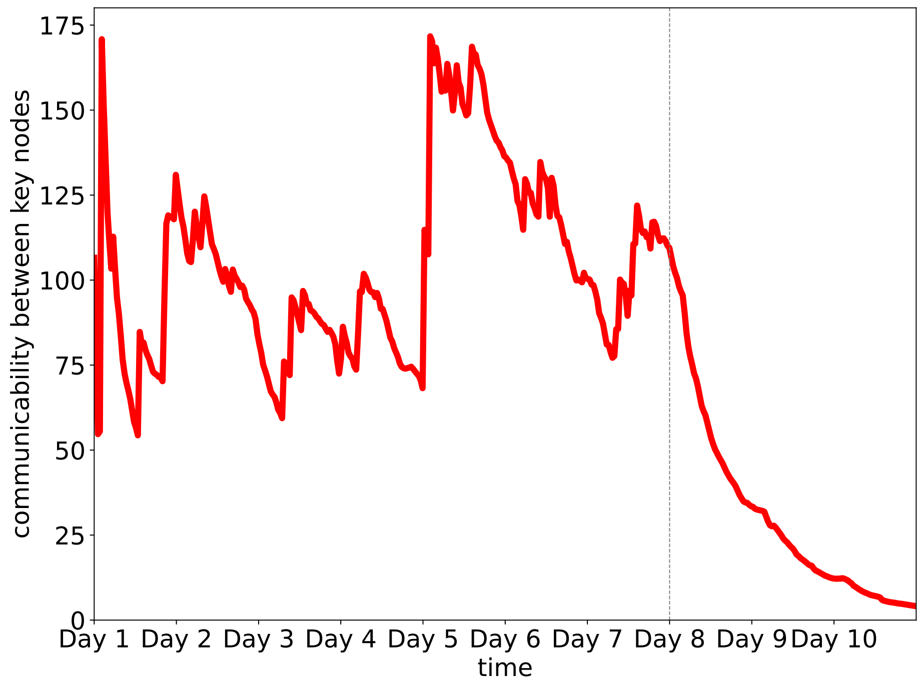


Figure 3.7: Voice call data: dynamic communicability between the five key nodes as a function of time.

Chapter 4

Conclusions

This research describes and analyzes a novel approach, using a continuous-time framework, to analyze dynamic network centrality, introduced in [10]. This new model is an improvement over existing methods that rely on analyzing individual snapshots of the network, as it allows for better data-driven simulations and theoretical analysis:

(i) The continuous-time framework fits better and in a more natural way with human communication patterns, allowing for a more accurate and realistic representation of communication dynamics. This is because it eliminates the need to divide the network data into predetermined time intervals, which can result in inaccuracies if they are too large or if, on the contrary, they are too finely spaced, in redundant computational processes or a false impression of accuracy.

(ii) By using ready-made, advanced numerical ODE solvers, network simulations can be carried out in a manner that time discretization is performed automatically ”under the hood”, ensuring high accuracy and efficiency. This approach enables us to effectively handle sudden and significant changes in network behavior in an adaptive manner.

(iii) Real-time summaries of centrality rankings can be monitored through this new ODE-based framework due to its property of downweighting information over time without the need to store or take account of all previous node interaction history.

(iv) Computing the dynamic receive centrality, $\mathbf{r}(t)$, is a factor N cheaper than dynamic broadcast centrality, $\mathbf{b}(t)$, in terms of storage and computational cost for sparse networks.

(v) The continuous-time framework is shown to be effective in real-time computation of dynamic centrality measures through the numerical experiments analyzed. These allowed us to illustrate that this type of measures can offer a better perspective of centrality than static or aggregate measures. Moreover, by tracking the most relevant nodes once they are known, this model is able to detect changes in the network structure.

Chapter 5

Further studies

Further generalization of the framework on dynamical systems at lower dimensions

A natural continuation of this study could be applied to different types of dynamical systems where only a few nodes compared to the total number of nodes are really significant in the behavior and evolution of such systems [10]. For instance, consider an evolving network $\mathbf{A}(t)$ with $N \times N$ dimensions, where N is very large. If a smaller subset of $M \ll N$ nodes is identified as significant, it might be worthwhile to explore an ordinary differential equation involving $\mathbf{V}(t) \in \mathbb{R}^{M\times M}$ with $M \times M$ dimensions. The equation could be in the form of

$$\mathbf{V}'(t) = P(\mathbf{V}(t)) + Q(\mathbf{V}(t))F(\mathbf{A}(t)),$$

where P and Q are polynomial or matrix-valued functions, and $F : [0,1]^{N\times N} \to \mathbb{R}^{M\times M}$ is an appropriate matrix-valued mapping. This kind of system would allow us to reduce the interactions among all N nodes to the subset of interest, and then measure the resulting changes in behavior in this lower dimension.

Potential use of the framework in applied scientific fields

Further studies could explore the potential of centrality measures such as the generalized Katz measure derived from this study and dynamic network broadcast/receive analysis in applied fields like fluid dynamics, atmospheric science or engineering:

- These measures could be used to study the complex spatio-temporal dynamics of turbulent flows, and identify key locations or structures that could be targeted for control or optimization.
- They can also be a useful tool in the study of fluid combustion, by identifying the critical points in the combustion system where the combustion reaction is most likely to be af-

fected by various factors such as temperature, pressure, and turbulence. This information can be used later to optimize the combustion process and improve its efficiency.

Many other scientific areas are suitable to the application of this novel framework such as social networks, traffic flow, transportation or power grids, neural networks, etc. since a huge number of real problems can be modeled through the use of evolving networks.

Overall, the application of centrality measures has the potential to provide valuable insights into the behavior of complex systems in multiple and diverse fields, enabling the development of more effective control and optimization strategies.

References

[1] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Mod. Phys.*, vol. 74, pp. 47–97, 2002. DOI: 10.1103/RevModPhys.74.47.

[2] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, pp. 39–43, 1953. DOI: 10.1007/BF02289026.

[3] M. Newman, *Networks*. Oxford, UK: Oxford University Press, 2018.

[4] F. Arrigo, D. J. Higham, V. Noferini, and R. Wood, "Dynamic katz and related network measures," *Linear Algebra and its Applications*, vol. 655, pp. 159–185, 2022. DOI: 10.1016/j.laa.2022.08.022.

[5] C. D. Meyer, *Matrix analysis and applied linear algebra*. Philadelphia, PA: SIAM, 2000.

[6] 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. DOI: 10.1016/S0169-7552(98)00110-X.

[7] H. Jeong, S. P. Mason, A.-L. Barabási, and Z. N. Oltvai, "Lethality and centrality in protein networks," *Nature*, vol. 411, no. 6833, pp. 41–42, 2001. DOI: 10.1038/35075138.

[8] S. Wasserman and K. Faust, *Social network analysis: Methods and applications*. Cambridge, UK: Cambridge University Press, 1994.

[9] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research*, vol. 33, no. 4, pp. 452–473, 1977. DOI: 10.1086/jar.33.4.3629752.

[10] P. Grindrod and D. J. Higham, "A dynamical systems view of network centrality," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 470, p. 20 130 835, 2014. DOI: 10.1098/rspa.2013.0835.

[11] P. Grindrod, M. C. Parsons, D. J. Higham, and E. Estrada, "Communicability across evolving networks," *Phys. Rev. E*, vol. 83, p. 046 120, 2011. DOI: 10.1103/PhysRevE.83.046120.

[12] P. Grindrod and D. J. Higham, “A matrix iteration for dynamic network summaries,” *SIAM Review*, vol. 55, no. 1, pp. 118–128, 2013. DOI: 10.1137/110855715.

[13] N. J. Higham, *Functions of matrices: theory and computation*. Philadelphia, PA: SIAM, 2008.

[14] N. Bilton, *The lifespan of a link*, http://bits.blogs.nytimes.com/2011/09/07/the-lifespan-of-a-link/?_php=true&_type=blogs&_r=0, [Online; accessed 7-March-2023], 2011.

[15] G. Grinstein, C. Plaisant, S. Laskowski, T. O’Connell, J. Scholtz, and M. Whiting, “Vast 2008 challenge: Introducing mini-challenges,” in *2008 IEEE Symposium on Visual Analytics Science and Technology*, New York, NY: IEEE, 2008, pp. 195–196. DOI: 10.1109/VAST.2008.4677383.

Appendix A

Python code

First synthetic experiment

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

##############################################################
################### SYNTHETIC EXPERIMENT 1 ##################
##############################################################

########################## MODULES ##########################

import numpy as np
from scipy.integrate import solve_ivp
from scipy.linalg import logm
import matplotlib.pyplot as plt

######################### VARIABLES #########################
# number of nodes
N = 31
# alpha and beta parameters
a = 0.7
b = 0.1
# depth of binary tree (2 even and 2 odd)
levels = 2
# identity matrix
I = np.eye(N)

######################### FUNCTIONS #########################

def unirandom(matrix):
    '''
    Introduces a 1 in a uniform randomized number of entries, specified by the
    parameter 'num_entries' in a given matrix.

            Parameters:
                    matrix (arr): matrix of size MxN

            Returns:
                    -
    '''
    # number of entries to randomize, uniform dist. with avg.= 5
    num_entries = int(np.random.uniform(low=0, high=10))

    if matrix.size < num_entries:
        raise ValueError("Invalid number of entries to randomize")

    # randomly select num_entries unique positions in the matrix
```

```
    positions = np.random.choice(matrix.size, size=num_entries, replace=False)
    # set the selected positions to 1
    matrix.flat[positions] = 1.

def A_even():
    '''
    Returns the constant value that takes A(t) at even time intervals with
    some noise added by 'unirandom' function.

            Parameters:
                    -

            Returns:
                    ematrix (arr): NxN matrix for even time intervals
    '''
    ematrix = np.zeros((N, N))
    for i in (2**k for k in range(0, levels*2 , 2)):
        for j in range(1, i+1):
            ematrix[i-1][i*2-1] = 1
            ematrix[i-1][i*2] = 1
            i = i + 1
    unirandom(ematrix)
    return ematrix

def A_odd():
    '''
    Returns the constant value that takes A(t) at odd time intervals with
    some noise added by 'unirandom' function.

            Parameters:
                    -

            Returns:
                    omatrix (arr): NxN matrix for odd time intervals
    '''
    omatrix = np.zeros((N, N))
    for i in (2**k for k in range(1, levels*2 , 2)):
        for j in range(1, i+1):
            omatrix[i-1][i*2-1] = 1
            omatrix[i-1][i*2] = 1
            i = i + 1
    unirandom(omatrix)
    return omatrix

def aggregate_out_degree(matrix):
    '''
    Returns the row sums that represent the aggregate out degree for each
    node (row) given an adjacency matrix.

            Parameters:
                    matrix (arr): adjacency matrix of size NxN

            Returns:
                    row_sums (list): list of agg. out degree for each node
    '''
    row_sums = [sum(row) for row in matrix]

    return row_sums


######################## A(t) ########################

# time interval t=[0, 20], with A(t) constant over each subinterval [i, i + 1)
# where A(t) = A_even and A(t) = A_odd when 'i' is even and odd respectively

A0 = A_even()
A1 = A_odd()
A2 = A_even()
A3 = A_odd()
A4 = A_even()
A5 = A_odd()
A6 = A_even()
```

```
A7 = A_odd()
A8 = A_even()
A9 = A_odd()
A10 = A_even()
A11 = A_odd()
A12 = A_even()
A13 = A_odd()
A14 = A_even()
A15 = A_odd()
A16 = A_even()
A17 = A_odd()
A18 = A_even()
A19 = A_odd()
A20 = np.zeros((N, N))

w0, v0 = eig(A0)
w1, v1 = eig(A1)
w2, v2 = eig(A2)
w3, v3 = eig(A3)
w4, v4 = eig(A4)
w5, v5 = eig(A5)
w6, v6 = eig(A6)
w7, v7 = eig(A7)
w8, v8 = eig(A8)
w9, v9 = eig(A9)
w10, v10 = eig(A10)
w11, v11 = eig(A11)
w12, v12 = eig(A12)
w13, v13 = eig(A13)
w14, v14 = eig(A14)
w15, v15 = eig(A15)
w16, v16 = eig(A16)
w17, v17 = eig(A17)
w18, v18 = eig(A18)
w19, v19 = eig(A19)

print("w0 =", w0)
print("w1 =", w1)
print("w2 =", w2)
print("w3 =", w3)
print("w4 =", w4)
print("w5 =", w5)
print("w6 =", w6)
print("w7 =", w7)
print("w8 =", w8)
print("w9 =", w9)
print("w10 =", w10)
print("w11 =", w11)
print("w12 =", w12)
print("w13 =", w13)
print("w14 =", w14)
print("w15 =", w15)
print("w16 =", w16)
print("w17 =", w17)
print("w18 =", w18)
print("w19 =", w19)

# aggregate matrix to compute the agg. out degree
agg_matrix = A0 + A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8 + A9 + A10 +\
             A11 + A12 + A13 + A14 +A15 + A16 + A17 + A18 + A19


# Adjacency matrix for t=[0,20]
def Adj(t):
    if 0.0 <= t < 1.0:
        return A0
    if 1.0 <= t < 2.0:
        return A1
    if 2.0 <= t < 3.0:
        return A2
    if 3.0 <= t < 4.0:
        return A3
```

```
    if 4.0 <= t < 5.0:
        return A4
    if 5.0 <= t < 6.0:
        return A5
    if 6.0 <= t < 7.0:
        return A6
    if 7.0 <= t < 8.0:
        return A7
    if 8.0 <= t < 9.0:
        return A8
    if 9.0 <= t < 10.0:
        return A9
    if 10.0 <= t < 11.0:
        return A10
    if 11.0 <= t < 12.0:
        return A11
    if 12.0 <= t < 13.0:
        return A12
    if 13.0 <= t < 14.0:
        return A13
    if 14.0 <= t < 15.0:
        return A14
    if 15.0 <= t < 16.0:
        return A15
    if 16.0 <= t < 17.0:
        return A16
    if 17.0 <= t < 18.0:
        return A17
    if 18.0 <= t < 19.0:
        return A18
    if 19.0 <= t <= 20.0:
        return A19
    return A20

######################## ODE ########################

# Matrix ODE function in vector form
def f(t, U):

    # Reshape U from vector to matrix
    U = U.reshape((N, N))

    # Compute the matrix ODE
    dUdt = -b * (U - I) - U @ logm(I - a * Adj(t))

    # Reshape dUdt from matrix to vector
    dUdt = dUdt.flatten()
    return dUdt


# Initial condition
U0 = np.eye(N)
U0 = U0.flatten()
#time span
t_span = (0, 20)

# Solve the matrix ODE numerically using Runge-Kutta 45 method
sol = solve_ivp(f, t_span, U0, method='RK45')

# Print the solution at discrete time points
print("\nsol_t :\n", sol.t)
print("\nsol_y :\n", sol.y)

# Communicability matrix U at t = 20 (entries >= 0)
U_t_20 = np.abs(sol.y[:,-1].reshape(N,N))

# broadcast centrality at t = 20
b_t_20 = U_t_20 @ np.ones(N)

# aggregate out degree
out_degree_list = aggregate_out_degree(agg_matrix)

```

```
print("\nb(t=20) :\n", b_t_20)
print("\nAgg. out degree =", out_degree_list)

######################### PLOTS #########################

fig, ax = plt.subplots()
ax.plot(list(range(1,N+1)), b_t_20, '*')
ax.set_title(r"First synthetic experiment: $\alpha=0.7$ $\beta=0.1$")
ax.set_xlabel("node")
ax.set_ylabel("dynamic broadcast  b(t=20)")

fig.savefig('exp1_bt20.eps', format='eps')

fig2, ax2 = plt.subplots()
ax2.plot(list(range(1,N+1)), out_degree_list, '*', color='r')
ax2.set_title("First synthetic experiment")
ax2.set_xlabel("node")
ax2.set_ylabel("aggregate out degree")

fig2.savefig('exp1_agg_out_degree.eps', format='eps')
```

Listing A.1: First synthetic experiment

Second synthetic experiment

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

#############################################################
################### SYNTHETIC EXPERIMENT 2 ##################
#############################################################

########################## MODULES ##########################

import numpy as np
from scipy.integrate import solve_ivp
from scipy.linalg import logm
import matplotlib.pyplot as plt

######################## VARIABLES #####################
# number of nodes
N = 17
# alpha and beta parameters
a = 0.9
b = 0.1

######################### A(t) #########################

# identity matrix of size NxN
I = np.eye(N)

# A(ti) for each interval where
# ti := [(i − 1)  , (i − 1 + 0.9)  ), for i = 0, 1, . . . , 7, and  = 0.1
# over the full interval t = [0, 3.5]

A0 = np.zeros((N, N)) # intercycle value, no connections

A1 = np.zeros((N, N))
A1[0][2] = 1
A1[2][0] = 1
A1[1][16] = 1
A1[16][1] = 1

A2 = np.zeros((N, N))
A2[2][3] = 1
A2[3][2] = 1

A3 = np.zeros((N, N))
```

```
A3[2][4] = 1
A3[4][2] = 1
A3[3][5] = 1
A3[5][3] = 1

A4 = np.zeros((N, N))
A4[0][16] = 1
A4[16][0] = 1
A4[1][2] = 1
A4[2][1] = 1
A4[6][3] = 1
A4[3][6] = 1
A4[4][7] = 1
A4[7][4] = 1
A4[5][9] = 1
A4[9][5] = 1

A5 = np.zeros((N, N))
A5[4][8] = 1
A5[8][4] = 1
A5[5][10] = 1
A5[10][5] = 1
A5[6][11] = 1
A5[11][6] = 1
A5[7][13] = 1
A5[13][7] = 1

A6 = np.zeros((N, N))
A6[6][12] = 1
A6[12][6] = 1
A6[7][14] = 1
A6[14][7] = 1
A6[8][15] = 1
A6[15][8] = 1

A7 = np.zeros((N, N))
A7[8][16] = 1
A7[16][8] = 1

# Adjacency matrix during five cycles
def Adj(t):
    if 0.0 <= t < 0.09:
        return A1
    if 0.1 <= t < 0.19:
        return A2
    if 0.2 <= t < 0.29:
        return A3
    if 0.3 <= t < 0.39:
        return A4
    if 0.4 <= t < 0.49:
        return A5
    if 0.5 <= t < 0.59:
        return A6
    if 0.6 <= t < 0.69:
        return A7

    if 0.8 <= t < 0.89:
        return A1
    if 0.9 <= t < 0.99:
        return A2
    if 1.0 <= t < 1.09:
        return A3
    if 1.1 <= t < 1.19:
        return A4
    if 1.2 <= t < 1.29:
        return A5
    if 1.3 <= t < 1.39:
        return A6
    if 1.4 <= t < 1.49:
        return A7

    if 1.6 <= t < 1.69:
```

```
            return A1
        if 1.7 <= t < 1.79:
            return A2
        if 1.8 <= t < 1.89:
            return A3
        if 1.9 <= t < 1.99:
            return A4
        if 2.0 <= t < 2.09:
            return A5
        if 2.1 <= t < 2.19:
            return A6
        if 2.2 <= t < 2.29:
            return A7

        if 2.4 <= t < 2.49:
            return A1
        if 2.5 <= t < 2.59:
            return A2
        if 2.6 <= t < 2.69:
            return A3
        if 2.7 <= t < 2.79:
            return A4
        if 2.8 <= t < 2.89:
            return A5
        if 2.9 <= t < 2.99:
            return A6
        if 3.0 <= t < 3.09:
            return A7

        if 3.2 <= t < 3.29:
            return A1
        if 3.3 <= t < 3.39:
            return A2
        if 3.4 <= t < 3.49:
            return A3
        if 3.5 <= t < 3.59:
            return A4
        if 3.6 <= t < 3.69:
            return A5
        if 3.7 <= t < 3.79:
            return A6
        if 3.8 <= t < 3.89:
            return A7

    return A0

######################### ODE #########################

# Matrix ODE function in vector form
def f(t, U):

    # Reshape U from vector to matrix
    U = U.reshape((N, N))

    # Compute the matrix ODE
    dUdt = -b * (U - I) - U @ logm(I - a * Adj(t))
    # Reshape dUdt from matrix to vector
    dUdt = dUdt.flatten()
    return dUdt


# Initial condition
U0 = np.eye(N)
U0 = U0.flatten()
# time span
t_span = (0, 3.5) # time span

# Solve the matrix ODE numerically using solve_ivp
sol = solve_ivp(f, t_span, U0)

# Print the solution at discrete time points
print("\nsol_t :\n", sol.t)
```

```
print("\nsol_y :\n", sol.y)

# broadcast centrality of nodes A (node 0) and B (node 1)
b_t_A = [(np.abs(sol.y[:,i].reshape(N,N)) @ np.ones(N))[0] for i in range(0,sol.y.shape[1])]
b_t_B = [(np.abs(sol.y[:,i].reshape(N,N)) @ np.ones(N))[1] for i in range(0,sol.y.shape[1])]

print("\nb(t)_A :\n", b_t_A)
print("\nb(t)_B :\n", b_t_B)

######################### PLOT #########################

fig, ax = plt.subplots()
ax.plot(sol.t, b_t_A, 'r', sol.t, b_t_B, 'b--')
ax.set_title(r"Second synthetic experiment: $\alpha=0.9$, $\beta=0.1$")
ax.set_xlabel("time (s)")
ax.set_ylabel("broadcast centrality  b(t)")
ax.legend(["b(t) of node A", "b(t) of node B"])
#ax.grid()
ax.set_xlim(0, 3.5)
ax.set_ylim(1, None)

fig.savefig('exp2_btA_vs_btB.eps', format='eps')
```

Listing A.2: Second synthetic experiment

Voice call experiment

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

##############################################################
################### VOICE CALL EXPERIMENT  ##################
##############################################################

########################## MODULES ##########################

import numpy as np
from scipy.integrate import solve_ivp
#from scipy.linalg import logm
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
import pandas as pd
from itertools import combinations
import time

######################## VARIABLES #####################
# number of nodes
N = 400
# alpha and beta parameters
a = 1E-4
b = 1.2E-5
# identity matrix of size NxN
I = np.eye(N)

# Path to the csv file
csv_file_path = "CellPhoneCallRecords.csv"

# Max. call time duration in seconds
t_max = 3000

################### FUNCTIONS #################

def sort_indexes(arr):
    '''
    Returns array indexes sorted in descending order from max value to min
    value.

            Parameters:
```

```
                    arr (arr): array to be sorted

            Returns:
                    sorted_indexes (arr): sorted array of indexes
    '''
    sorted_indexes = sorted(range(len(arr)), key=lambda i: arr[i], reverse=True)
    return sorted_indexes

def symmetric_sum(matrix):
    '''
    Returns resulting matrix from the sum of symmetric elements of a matrix.

            Parameters:
                    matrix (arr): NxN array

            Returns:
                    res (arr): NxN array with sum of symmetric entries
    '''
    n = len(matrix)
    res = np.zeros((n,n))
    for i in range(n):
        for j in range(i, n):
            if i == j:
                res[i][j] = matrix[i][j]
            else:
                res[i][j] = matrix[i][j] + matrix[j][i]
                res[j][i] = res[i][j]
    return res

#################### logm approx. #####################

def logm_approx(A, a=a):
    '''
    Returns the approximation of the function logm by truncation to the
    fifth power of its series expansion.

            Parameters:
                    A (arr): NxN array
                    a (float): parameter

            Returns:
                    principal matrix logarithm approximation
    '''
    A2 = A @ A
    A3 = A2 @ A
    A4 = A3 @ A
    A5 = A4 @ A

    return a * A - (a**2/2) * A2 + (a**3/3) * A3 - (a**4/4) * A4 + (a**5/5) * A5


#### Dynamic communicability of key nodes ####

def dyncomm_key_nodes(U):

    # Calculate the sum and average of all entries except the main diagonal
    total_sum = np.sum(U) - np.sum(np.diag(U))
    total_avg = total_sum / (N*N - N)

    # key nodes indexes
    nodes = [1, 2, 3, 5, 200]

    # All pairwise combinations
    pairs = list(combinations(nodes, 2))

    # key nodes sum
    key_sum = 0
    for item in pairs:
        key_sum += U[item[0]][item[1]]
        key_sum += U[item[1]][item[0]]

    # key nodes average
```

```python
    key_sum_avg = key_sum / (len(pairs) * 2)

    if total_avg == 0:
        return 0

    return key_sum_avg / total_avg


################## AGGREGATE BANDWIDTH #################
# bandwidth of a node is defined as the aggregate time call duration in which
# this node is active over a certain period

# Read the csv file into a Pandas DataFrame
df = pd.read_csv(csv_file_path)

# Convert the Datetime field to a datetime object
df["Datetime"] = pd.to_datetime(df["Datetime"], format="%Y%m%d %H%M")

# Filter the data for days 1 to 10
start_date = pd.Timestamp(2006, 6, 1)
end_date = pd.Timestamp(2006, 6, 10, 23, 59, 59)

df = df[(df["Datetime"] >= start_date) & (df["Datetime"] <= end_date)]

# Group the data by 'From' and 'To' fields and sum the call durations
grouped = df.groupby(["From", "To"])["Duration(seconds)"].sum()

# Initialize the matrix for call time duration for each node with zeros
agg_matrix = np.zeros((N, N))

# Update the matrix entries with the sum of call durations
for (from_cell, to_cell), duration in grouped.items():
    agg_matrix[from_cell][to_cell] = duration

# sum symmetric elements (node as sender or receiver)
agg_matrix = symmetric_sum(agg_matrix)

# aggregate vector with the bandwidth of each node from day 1 to 10
agg_day1_to_10 = [np.sum(row) for row in agg_matrix]

print("agg_day1_to_10: (first ten elem.): \n", agg_day1_to_10[0:10])

######################### A(t) #########################

# Read the csv file into a Pandas DataFrame
df = pd.read_csv(csv_file_path)

# Convert the Datetime field to a datetime object
df["Datetime"] = pd.to_datetime(df["Datetime"], format="%Y%m%d %H%M")

def Adj(t):
    '''
    Returns the adjacency matrix of the network at a given time 't'.
    Sets elements to 1 if 't' belongs to the datetime conversation which nodes
    are given in the .csv file by the fields 'From' and 'To'.

            Parameters:
                    t (float): time

            Returns:
                    A (arr): NxN adjacency matrix
    '''

    # Filter all datetimes in the interval actual time - max calltime duration
    actual_time = t0_datetime + pd.Timedelta(seconds=t)
    start_time = actual_time - pd.Timedelta(seconds=t_max)

    filtered = df[(df["Datetime"] >= start_time) & (df["Datetime"] <= actual_time)]

    # Initialize the adjacency matrix with zeros
    A = np.zeros((N, N))

```

```
    # Set matrix entries to 1 if the time is inside a conversation
    for index, row in filtered.iterrows():
        from_cell = row["From"]
        to_cell = row["To"]
        start_call_time = row["Datetime"]
        end_call_time = start_call_time + pd.Timedelta(seconds=row["Duration(seconds)"])
        if end_call_time >= actual_time:
            A[from_cell][to_cell] = 1
            A[to_cell][from_cell] = 1

    return A

######################## Alternative Adj(t)####################

# Read the csv file into a Pandas DataFrame
#df = pd.read_csv(csv_file_path)

# Convert the Datetime field to a datetime object
#df["Datetime"] = pd.to_datetime(df["Datetime"], format="%Y%m%d %H%M")

# Extract the start and end time of each call
#df['start_call'] = df['Datetime']
#df['end_call'] = df['Datetime'] + pd.to_timedelta(df['Duration(seconds)'], unit='s')

def Adj2(t):

    # Filter the dataframe to keep only the active calls at time 't'
    actual_time = t0_datetime + timedelta(seconds=t)
    active_calls = df[(actual_time >= df['start_call']) & (actual_time <= df['end_call'])]

    # Create the adjacency matrix
    A = np.zeros((N, N))
    for _, row in active_calls.iterrows():
        A[row['From'], row['To']] = 1
        A[row['To'], row['From']] = 1

    return A

######################### ODE ########################

# Matrix ODE function in vector form
def f(t, U):

    # Reshape U from vector to matrix
    U = U.reshape((N, N))

    # Compute the matrix ODE
    dUdt = -b * (U - I) - U @ logm_approx(Adj(t))
    # Reshape dUdt from matrix to vector
    dUdt = dUdt.flatten()
    return dUdt

######################### day 1 to 7 broadcast ########################

start_time = time.time()

# Initial condition
U0 = np.eye(N)
U0 = U0.flatten()

# Time interval
t0_datetime = pd.Timestamp(2006, 6, 1)
tf_datetime = pd.Timestamp(2006, 6, 6, 23, 59 ,59)
delta = tf_datetime - t0_datetime

# Time span
t0 = 0
tf = delta.total_seconds()
t_span = (t0, tf)

# Solve the matrix ODE numerically using solve_ivp
sol = solve_ivp(f, t_span, U0, method='RK23', rtol=1e-4, atol=1e-4)
```

```

# Print the solution at discrete time points
#print("\nsol_t :\n", sol.t)
#print("\nsol_y :\n", sol.y)

# Communicability matrix U at t=tf (day 7)
U_t_7 = np.abs(sol.y[:,-1].reshape(N,N))

# broadcast centrality at t = tf (day 7)
b_t_7 = U_t_7 @ np.ones(N)

print("\nb(t='day 7') :\n", b_t_7)

print("\nb_t_day7[1] =", b_t_7[1])
print("b_t_day7[2] =", b_t_7[2])
print("b_t_day7[3] =", b_t_7[3])
print("b_t_day7[5] =", b_t_7[5])
print("b_t_day7[200] =", b_t_7[200])

#print("\nsorted_b_indexes(t='day 7') : ",sort_indexes(b_t_7)[0:5])

# receive centrality at t = tf (day 7)
r_t_7 = U_t_7.T @ np.ones(N)

print("\nBroadcast from day 1 - 7 done!")

end_time = time.time()
total_time = end_time - start_time

print("Time taken:", round(total_time, 2), "seconds")

######################## day 7 to 10 broadcast #########################

start_time = time.time()

# Initial condition
U0 = np.eye(N)
U0 = U0.flatten()

# Time interval
t0_datetime = pd.Timestamp(2006, 6, 7)
tf_datetime = pd.Timestamp(2006, 6, 10, 23, 59, 59)
delta = tf_datetime - t0_datetime

# Time span
t0 = 0
tf = delta.total_seconds()
t_span = (t0, tf)

# Solve the matrix ODE numerically using solve_ivp
sol = solve_ivp(f, t_span, U0, method='RK23', rtol=1e-4, atol=1e-4)

# Print the solution at discrete time points
#print("\nsol_t :\n", sol.t)
#print("\nsol_y :\n", sol.y)

# Communicability matrix U at t=tf (day 10)
U_t_10 = np.abs(sol.y[:,-1].reshape(N,N))

# broadcast centrality at t = tf (day 10)
b_t_10 = U_t_10 @ np.ones(N)

print("\nb(t='day 10') :\n", b_t_10)

print("\nb_t_day10[309] =", b_t_10[309])
print("b_t_day10[392] =", b_t_10[392])
print("b_t_day10[360] =", b_t_10[360])
print("b_t_day10[306] =", b_t_10[306])
print("b_t_day10[300] =", b_t_10[300])

#print("\nsorted_b_indexes(t='day 10') : ",sort_indexes(b_t_10)[0:5])

```

```
# Receive centrality at t = tf (day 10)
r_t_10 = U_t_10.T @ np.ones(N)

print("\nBroadcast from day 7 - 10 done!")

end_time = time.time()
total_time = end_time - start_time

print("Time taken:", round(total_time, 2), "seconds")

######################### PLOTS #########################

fig, ax = plt.subplots(figsize=(10,8))
ax.plot(agg_day1_to_10, b_t_7, '+', color='blue', label="rest of nodes")
ax.plot(agg_day1_to_10[200], b_t_7[200], 'v', color='red', label="ring leader ID before day 7")
ax.plot(agg_day1_to_10[1], b_t_7[1], 's', color='red', label="key nodes IDs before day 7")
ax.plot(agg_day1_to_10[2], b_t_7[2], 's', color='red')
ax.plot(agg_day1_to_10[3], b_t_7[3], 's', color='red')
ax.plot(agg_day1_to_10[5], b_t_7[5], 's', color='red')
ax.plot(agg_day1_to_10[300], b_t_7[300], '^', color='green', label="ring leader ID after day 7")
ax.plot(agg_day1_to_10[309], b_t_7[309], 'd', color='green', label="key nodes IDs after day 7")
ax.plot(agg_day1_to_10[392], b_t_7[392], 'd', color='green')
ax.plot(agg_day1_to_10[360], b_t_7[360], 'd', color='green')
ax.plot(agg_day1_to_10[306], b_t_7[306], 'd', color='green')
ax.set_title("Voice call experiment")
ax.set_xlabel("bandwidth")
ax.set_ylabel("broadcast centrality at day 7")
ax.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
ax.set_xlim(0, 3.5e5)
ax.legend()

fig.savefig('voicecall_exp_1_7.eps', format='eps')

fig2, ax2 = plt.subplots(figsize=(10,8))
ax2.plot(agg_day1_to_10, b_t_10, '+', color='blue', label="rest of nodes")
ax2.plot(agg_day1_to_10[200], b_t_10[200], 'v', color='red', label="ring leader ID before day 7")
ax2.plot(agg_day1_to_10[1], b_t_10[1], 's', color='red', label="key nodes IDs before day 7")
ax2.plot(agg_day1_to_10[2], b_t_10[2], 's', color='red')
ax2.plot(agg_day1_to_10[3], b_t_10[3], 's', color='red')
ax2.plot(agg_day1_to_10[5], b_t_10[5], 's', color='red')
ax2.plot(agg_day1_to_10[300], b_t_10[300], '^', color='green', label="ring leader ID after day 7"
    )
ax2.plot(agg_day1_to_10[309], b_t_10[309], 'd', color='green', label="key nodes IDs after day 7")
ax2.plot(agg_day1_to_10[392], b_t_10[392], 'd', color='green')
ax2.plot(agg_day1_to_10[360], b_t_10[360], 'd', color='green')
ax2.plot(agg_day1_to_10[306], b_t_10[306], 'd', color='green')
ax2.set_title("Voice call experiment")
ax2.set_xlabel("bandwidth")
ax2.set_ylabel("broadcast centrality at day 10 (from day 7)")
ax2.ticklabel_format(axis="x", style="sci", scilimits=(0,0))
ax2.set_xlim(0, 3.5e5)
ax2.legend()

fig2.savefig('voicecall_exp_7_10.eps', format='eps')

######################### day 1 to 10 broadcast #########################

# Initial condition
U0 = np.eye(N)
U0 = U0.flatten()

# Time interval
t0_datetime = pd.Timestamp(2006, 6, 1)
tf_datetime = pd.Timestamp(2006, 6, 11)
delta = tf_datetime - t0_datetime

# Time span
t0 = 0
tf = delta.total_seconds()
t_span = (t0, tf)

# Solve the matrix ODE numerically using solve_ivp
```

```
sol = solve_ivp(f, t_span, U0, method='RK45', atol=1e-4, rtol=1e-4)

# dynamic communicability vector
comm_1_10 = np.zeros(sol.y.shape[1])

for i in range(sol.y.shape[1]):
    comm_1_10[i] = dyncomm_key_nodes(sol.y[:,i].reshape(N,N))


# No communication in the first 8 min, removing zero communicability
# Boolean mask indicating which elements are close to zero
mask = np.isclose(comm_1_10, 0, atol=1e-5)

# Filtered communicability array using the mask
filtered_comm = comm_1_10[~mask]

# Array without the close-to-zero elements
comm_1_10 = filtered_comm[np.nonzero(filtered_comm)]
sol.t = sol.t[np.nonzero(filtered_comm)]

print("communicability vector: \n", comm_1_10)

###### PLOT ######

fig3, ax3 = plt.subplots(figsize=(10,8))
x_label = np.arange(1, 11)
ax3.plot(np.linspace(1, 11, num = len(comm_1_10)), comm_1_10, color='red')
ax3.set_xlim(1, 11)
ax3.set_ylim(0, None)
ax3.set_xlabel("time")
ax3.set_ylabel("communicability between key nodes")
ax3.set_xticks(x_label)
ax3.set_xticklabels(['Day {}'.format(i) for i in x_label])
ax3.axvline(x=8, linestyle='--', color='gray', linewidth=0.8)

fig3.savefig('voicecall_exp_dyncomm.eps', format='eps')
```

Listing A.3: Voice call experiment

$\mathbf{b}(t)$ vs. $\mathbf{r}(t)$ cost comparison

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

#############################################################
#################### RECEIVE CENTRALITY ####################
#############################################################

######################### MODULES #########################

import numpy as np
from scipy.integrate import solve_ivp
#from scipy.linalg import logm
import pandas as pd
import time

####################### VARIABLES #####################
# number of nodes
N = 400
# alpha and beta parameters
a = 1E-4
b = 1.2E-5
# identity matrix of size NxN
I = np.eye(N)
# ones vector
r0 = np.ones(N)
# tolerances
RTOL = 0.05
```

```python
ATOL = 0.01

# Path to the csv file
csv_file_path = "CellPhoneCallRecords.csv"

# Max. call time duration in seconds
t_max = 3000

#################### logm approx. #####################

def logm_approx(A, a=a):
    '''
    Returns the approximation of the function logm by truncation to the
    fifth power of its series expansion.

            Parameters:
                    A (arr): NxN array
                    a (float): parameter

            Returns:
                    principal matrix logarithm approximation
    '''
    A2 = A @ A
    A3 = A2 @ A
    A4 = A3 @ A
    A5 = A4 @ A

    return a * A - (a**2/2) * A2 + (a**3/3) * A3 - (a**4/4) * A4 + (a**5/5) * A5

######################### A(t) #########################

# Read the csv file into a Pandas DataFrame
df = pd.read_csv(csv_file_path)

# Convert the Datetime field to a datetime object
df["Datetime"] = pd.to_datetime(df["Datetime"], format="%Y%m%d %H%M")

def Adj(t):
    '''
    Returns the adjacency matrix of the network at a given time 't'.
    Sets elements to 1 if 't' belongs to the datetime conversation which nodes
    are given in the .csv file by the fields 'From' and 'To'.

            Parameters:
                    t (float): time

            Returns:
                    A (arr): NxN adjacency matrix
    '''
    # Filter all datetimes in the interval actual time - max calltime duration
    actual_time = t0_datetime + pd.Timedelta(seconds=t)
    start_time = actual_time - pd.Timedelta(seconds=t_max)

    filtered = df[(df["Datetime"] >= start_time) & (df["Datetime"] <= actual_time)]

    # Initialize the adjacency matrix with zeros
    A = np.zeros((N, N))

    # Set matrix entries to 1 if the time is inside a conversation
    for index, row in filtered.iterrows():
        from_cell = row["From"]
        to_cell = row["To"]
        start_call_time = row["Datetime"]
        end_call_time = start_call_time + pd.Timedelta(seconds=row["Duration(seconds)"])
        if end_call_time >= actual_time:
            A[from_cell][to_cell] = 1
            A[to_cell][from_cell] = 1

    return A

######################### ODE #########################

```

```
# Matrix ODE functions
def f1(t, U):

    f1.num_calls += 1

    # Reshape U from vector to matrix
    U = U.reshape((N, N))

    # Compute the matrix ODE
    dUdt = -b * (U - I) - U @ logm_approx(Adj(t))
    # Reshape dUdt from matrix to vector
    dUdt = dUdt.flatten()
    return dUdt

def f2(t, r):

    f2.num_calls += 1

    # Compute the matrix ODE
    drdt = -b * (r - r0) - (logm_approx(Adj(t)).T) @ r
    return drdt

###############################################################

def method1(t0_datetime, tf_datetime):

    # Initial condition
    U0 = np.eye(N)
    U0 = U0.flatten()

    delta = tf_datetime - t0_datetime

    # Time span
    t0 = 0
    tf = delta.total_seconds()
    t_span = (t0, tf)
    t_eval = np.linspace(t0, tf, num=100)

    # Solve the matrix ODE numerically using solve_ivp
    sol = solve_ivp(f1, t_span, U0, method='RK45', t_eval=t_eval, atol = ATOL, rtol = RTOL)

    return sol


def method2(t0_datetime, tf_datetime):

    # Initial condition
    r0 = np.ones(N)

    delta = tf_datetime - t0_datetime

    # Time span
    t0 = 0
    tf = delta.total_seconds()
    t_span = (t0, tf)
    t_eval = np.linspace(t0, tf, num=100)

    # Solve the matrix ODE numerically using solve_ivp
    sol = solve_ivp(f2, t_span, r0, method='RK45', t_eval=t_eval, atol = ATOL, rtol = RTOL)
    return sol

######################## TIMES ########################

# Time interval
t0_datetime = pd.Timestamp(2006, 6, 1)
tf_datetime = pd.Timestamp(2006, 6, 1, 23, 59 ,59)

# number of repetions
rep = 100

total_calls = 0
total_time = 0
```

```
for i in range(rep):
    f1.num_calls = 0
    # Time taken by method 1
    start_time = time.time()
    sol_1 = method1(t0_datetime, tf_datetime)
    end_time = time.time()
    duration = end_time - start_time
    total_calls += f1.num_calls
    total_time += duration

print("###### Method 1: b(t)######")
print("total time =", total_time)
print("function calls = ", total_calls)
r1 = round((total_time / total_calls) * 1000, 2)
print(f"Ratio ms/call ({rep} rep.): {r1}\n")

total_calls = 0
total_time = 0

for i in range(rep):
    f2.num_calls = 0
    # Time taken by method 2
    start_time = time.time()
    sol_2 = method2(t0_datetime, tf_datetime)
    end_time = time.time()
    duration = end_time - start_time
    total_calls += f2.num_calls
    total_time += duration

print("###### Method 2: r(t) ######")
print("total time =", total_time)
print("function calls = ", total_calls)
r2 = round((total_time / total_calls) * 1000, 2)
print(f"Ratio ms/call ({rep} rep.): {r2}")

print("% improvement =", round(((r2 - r1) / r1) * 100, 2))
```

Listing A.4: $\mathbf{b}(t)$ vs. $\mathbf{r}(t)$ cost comparison

Bachelor’s Theses in Mathematical Sciences 2023:K10
ISSN 1654-6229

LUNFNA-4045-2023

Numerical Analysis
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/