University of Piraeus
Department of Digital Systems
Information Systems and Services
Big Data and Analytics

Community Detection in Signed Directed Graphs

MASTER THESIS

Manolis A. Androulidakis

Supervisor: Maria Halkidi, Associate Professor

Piraeus, February 2021

Acknowledgements

At this point I would like to thank my supervisor Maria Halkidi, Associate Professor in the Department of Digital Systems, University of Piraeus, for giving me the opportunity to work on this exciting project for my master thesis.

Also, I would like to thank my family and friends for supporting me throughout this process.

Manolis Androulidakis
February 2021

Abstract

Graph clustering is a fundamental technique that partitions similar nodes into clusters. It is of great importance in large-scale networks and is widely adopted in a variety of scientific areas where it is crucial to identify patterns or structures quickly, such as Social Network Analysis, Statistical Data Analysis, Data Mining, Machine Learning, Biology and others.

In this thesis, we focus on signed directed graphs and examine a community detection algorithm that groups together nodes with similar characteristics. The single most significant factor for efficient graph clustering is the computation of the similarity score among nodes. Our main challenge is to establish a similarity relationship among nodes based on the connectivity patterns they follow, i.e., combine the concepts positive and negative co-citation and co-reference.

First, we partition the original graph into its positive and negative subgraphs and work in parallel on both subgraphs. For each subgraph, we apply citation analysis and binding theory to calculate the co-citation and co-reference matrices, respectively. The former contains the number of nodes that two nodes both point to, while the latter gives the number of nodes that commonly point to two nodes. We use normalized mathematical models to smoothen data, factoring in the degree of each node. Further, to remove outliers, we tested taking into consideration the balance among positive and negative incoming and outgoing links among nodes. That is, the number of common links between two nodes counts less when these nodes share significantly less positive than negative out-links, and vice versa.

In this work, we also provide a complete implementation of the proposed algorithm in Python, as well as the datasets that we used to experimentally evaluate its correctness and accuracy. With this, we prove that the proposed algorithm returns the expected results for randomly generated signed directed graphs and scales up to thousands of nodes, depending on the density of the original graph.

Keywords

Statistical Data Analysis, Social Network Analysis, Graph Clustering, Community Detection, Similarity Score, Co-Citation, Co-Reference, Balance, Affinity Propagation, k-means, Python, NetworkX, numpy

Περίληψη

Η συσταδοποίηση σε γράφους είναι μια θεμελιώδης τεχνική που κατηγοριοποιεί παρόμοιους κόμβους ενός γράφου σε ομάδες (συστάδες). Είναι πολύ σημαντική για δίκτυα μεγάλης κλίμακας και χρησιμοποιείται ευρέως σε επιστημονικά πεδία όπου είναι απαραίτητη η ταυτοποίηση μοτίβων ή δομών, όπως η ανάλυση κοινωνικών δικτύων, η στατιστική Ανάλυση Δεδομένων, η Εξόρυξη Δεδομένων, η Μηχανική Μάθηση, η Βιολογία και άλλα.

Στη παρούσα διπλωματική επικεντρωνόμαστε σε κατευθυνόμενους γράφους και εξετάζουμε ένα αλγόριθμο ανίχνευσης κοινοτήτων, ο οποίος ομαδοποιεί κόμβους με παρεμφερή χαρακτηριστικά. Ο πιο σημαντικός παράγοντας για αποδοτική συσταδοποίηση είναι ο υπολογισμός του μέτρου ομοιότητας μεταξύ κόμβων. Η πιο σημαντική πρόκληση είναι ο καθορισμός της σχέσης ομοιότητας μεταξύ κόμβων, με βάση τα μοτίβα συνδεσιμότητας που ακολουθούν οι κόμβοι, δηλαδή ο συνδυασμός των εννοιών co-reference και co-citation.

Αρχικά, χωρίζουμε τον γράφο στο θετικό και αρνητικό υπο-γράφο του και δουλεύουμε παράλληλα και στους δύο υπο-γράφους. Για κάθε υπο-γράφο αναλύουμε τις αναφορές και τις συνδέσεις μεταξύ των κόμβων προκειμένου να υπολογίσουμε τους πίνακες co-citation και co-reference, αντίστοιχα. Ο πρώτος περιέχει τον αριθμό κόμβων στους οποίους αναφέρονται δύο κόμβοι, και ο δεύτερος δίνει τον αριθμό κόμβων που δείχνουν δύο κόμβοι. Χρησιμοποιούμε κανονικοποιημένα μαθηματικά μοντέλα για να εξομαλύνουμε τα δεδομένα, συν-υπολογίζοντας τον βαθμό κάθε κόμβου. Προκειμένου να απομακρυνθούν τα ακραία στοιχεία, επιχειρήσαμε να λάβουμε υπόψη την ισορροπία θετικών και αρνητικών εισερχόμενων και εξερχόμενων ακμών μεταξύ των κόμβων, ούτως ώστε ο αριθμός κοινών ακμών μεταξύ δύο κόμβων να μετράει λιγότερο όταν αυτοί οι κόμβοι μοιράζονται σημαντικά λιγότερες θετικές από αρνητικές ακμές και αντίστροφα.

Συνεχίζοντας, κάνουμε μια πλήρη υλοποίηση του αλγορίθμου σε python προκειμένου να ελέγξουμε πειραματικά την ορθότητα και την ακρίβεια του αλγορίθμου. Τέλος, παρουσιάζουμε τα πειραματικά αποτελέσματα του αλγορίθμου ο οποίος υλοποιείται σε signed, directed γράφους χιλιάδων κόμβων, ανάλογα με την πυκνότητα του γράφου.

Λέξεις-Κλειδιά

Στατιστική Ανάλυση Δεδομένων, Ανάλυση Κοινωνικών Δικτύων, Συσταδοποίηση Γράφων, Ανίχνευση Κοινοτήτων, Μέτρο Ομοιότητας, Ισορροπία, Affinity Propagation, k-means, Python, NetworkX, numpy

1 Introduction

1.1 Problem Statement

Discovering communities in a network is a fundamental problem in network science that gains increasing attention in the scientific community. In the era of big data, community identification and detection has proven to be an invaluable tool to manipulate and understand large-scale network data as it enables the study of mesoscopic structures that are often associated with behavioral and functional characteristics of the underlying networks.

The problem that community detection attempts to solve is the identification of vertices that share similar properties or characteristics. Detecting and analyzing how individual communities are structured leads to deeper understanding of the underlying phenomena that occur in these subsystems, and progressively, in the original system as a whole.

In this context, a major challenge when partitioning a large graph into smaller clusters is the definition of the similarity score among its nodes. Traditionally, clustering algorithms use a standard distance metric, such as Euclidean, Minkowski, Manhattan etc., to compare nodes and iteratively group them into clusters. However, to outline domain-specific semantics into the comparison one has to define a custom similarity measure among nodes, i.e., calculate a custom similarity matrix for the graph under study. For signed, directed networks this allows us to weigh in characteristics and attributes of nodes, i.e., introduce and combine the concepts of positive and negative co-citation and co-reference.

In general we can state that a network has a community structure if it is possible to group effectively the nodes of the network in such a way that every set of nodes is connected internally in a dense way. More specifically, in the special case of non-overlapping community identification, this means that nodes are divided in groups that are characterised internally with dense connections and the connections between groups are sparse. Overlapping communities can sometimes take place. In principle the definition of community clustering relates to the idea that pairs of nodes have a higher chance of being connected when they belong to the same group, and a lower chance of being connected if they belong to different groups. A problem that shares a lot of similar characteristics to the above mentioned one is community search, in which the main aim is to identify a group that a certain node belongs to.

In the study of networks, such as computer and information networks, social networks and biological networks, a number of different characteristics have been found to occur commonly, including the small-world property, heavy-tailed degree distributions, and clustering, among others. Another common characteristic is community structure. In the context of networks, community structure refers to the occurrence of groups of nodes in a network that are more densely connected internally than with the rest of the network, as shown in the example image to the right. This inhomogeneity of connections suggests that the network has certain natural divisions within it.

Defining communities effectively involves the division in a set of vertices in a way that every vertex belongs to one and only one community. Despite this simplification, which is widely used, a more complete representation could involve cases where vertices belong to more than one community. This is possible in social networks. In such cases every node depicts a person, while groups of people are presented as communities. Different types of communities could represent family, colleagues, friends from the same hobbies etc. Using community detection to study different groups of social relationships is quite common, and is an example of overlapping communities in the real world.

It is possible for certain networks to lack community structure. Actually in some basic network models, such as the Barabasi-Albert model, a community structure does not appear. In real networks on the other hand, community structure appears quite often.

1.2 Real-world use cases

The prevalence of digital technology and wide adoption of the Internet in most aspects of human lives has led to an exponential growth of data, which have already reached the order of zettabytes. A multitude of major research areas like sociology, biology and computer science gradually establish on graphs as the defacto standard to represent models and systems in a concise and cost-efficient manner. Over the years, this has elevated the importance of detecting communities in graphs which greatly affects decision making in both the scientific and business worlds.

Originally, the concept of communities was associated with individual networks of human actors that exhibited certain characteristic structural properties. However, in more complex, digital systems, the scope of communities broadens and does not necessarily involve human actors.

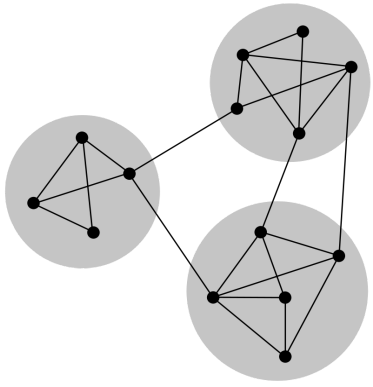


Figure 1: A sketch of a small network displaying **community structure**, with three groups of nodes with dense internal connections and sparser connections between groups.

Finding a community structure in a network is of crucial importance for several reasons. Communities allow us to create a large scale map of a network since individual communities act like meta-nodes in the network which makes its study easier.

Given that communities commonly are related to operational components of a system, identifying communities can illuminate the operation of the system as a whole. Concerning metabolic networks, these communities are related to different cycles and concerning protein interaction networks, communities are related to proteins that function closely inside biological cells. If biology is modular then clusters, or communities, of proteins derived using only protein interaction network structure should define protein modules with similar biological roles.

Similarly, in social network analysis, community detection is a basic step to understand the structure and function of networks.

Accordingly, citation networks form communities by research topic. The ability to detect sub-structures inside these graphs could possibly help explain how network operation and topology affect each other. The usefulness of such a procedure is very important since it could potentially improve algorithms on graphs.

One of the principal reasons why community detection is useful, is that these communities are commonly characterised by a different set of attributes and roles, when compared to the average attributes and roles of the networks. As such, ignoring communities is definitely going to lead to the omission of essential features of these networks.

Existence of communities also generally affects various processes like rumour spreading or epidemic spreading happening on a network. Hence to properly understand such processes, it is important to detect communities and also to study how they affect the spreading processes in various settings.

Finally, an important application that community detection has found in network science is the link prediction concerning cases where certain links are lacking, and the detection of links that could potentially be deemed false or abnormal in the network. It is possible, and actually quite common, that certain links may get omitted and it is equally common that abnormal links may get imported into the dataset, since errors can be potentially common during measurement. Cluster identification algorithms can handle both of these cases especially well since the assignment of probability of existence of an edge is allowed.

1.2.1 Bibliographic Citation

Bibliographic networks is a one of the most widely known categories where graph clustering algorithms apply. Analysing bibliographic networks is important for understanding the process of scientific publications. For example, there are various algorithms that operate on bibliographic graphs in order to achieve semantic topic learning and qualified community detection using both path-related features and node embeddings.

Other than the above, there is ongoing research on community detection in multi-relational bibliographic networks which incorporate the different types of objects and relationships. That is, various bibliographic networks are often modeled as Heterogeneous Information Networks (HINs). Mining HIN has become a hot research topic which attracts a lot

of attention due to its capability of capturing meta structures with various rich semantic meanings wide applications in real-world scenarios including recommender systems, clustering, and outlier detections.

Namely, authors can be connected by paper with term and venue information. The proximities between authors can be measured through mining if they coauthor a paper, publish a paper in the same venue, or publish paper with citations between them each of which may form a meta-path that refers to a different semantic state. (Zhou et al.) [1]

1.2.2 World Wide Web

The World-Wide Web has spawned a sharing and dissemination of information on an unprecedented scale. Hundreds of millions of individuals are creating, annotating, and exploiting hyperlinked content in a distributed fashion. These individuals come from a variety of backgrounds and have a variety of motives for creating the content. The hyperlinks of the Web give it additional structure; the network of these links is a rich source of latent information. (Kleinberg et al.) [2]

1.2.3 Social Networks

Proper social research starts from the premise that people are connected and not just atomized individuals. Nowadays, social networks like Facebook, YouTube and Twitter have reached huge sizes counting billions of nodes. A social network is usually represented by a graph consisting of a set of nodes and edges connecting these nodes. The nodes represent the individuals/entities, and the edges correspond to the interactions among them.

In social networks analysis communities are defined in a quite different way, i.e., by looking at how people are connected to each other, and clustering them into groups. The tendency of people with similar tastes, choices, and preferences to get associated in a social network leads to the formation of virtual clusters or communities. Thus, communities in social networks are a statistical measure of connectivity.

Understanding communities in social media networks is vital in controlling the way information spreads across different audiences. Different groups may well benefit from different messaging specifically targeted to their needs and interests.

The dynamic nature of social networks is a key factor that must be taken into account when examining social networks and their communities. As social networks evolve, self-organization, tendencies and connections usually change over time leading to different community structure. This adds another dimension in the analysis needed to observe behaviors of groups in social networks and make predictions or assumptions. (Nettleton et al.) [3]

1.2.4 Biology

Network analysis and modeling is a rapidly growing area which is moving forward our understanding of biological processes. In the past few decades there has been a vast increase in the amount of biological data that is available to the scientific community. As such, a whole new perspective has emerged in studying complex networks that are embedded in biological systems. For example, there are studies that focus on the evolution of networks at the gene and protein level and to the dynamics and stability of communities. One of the current challenges is to recognize the commonalities in evolutionary and ecological applications of network thinking to create a predictive science of biological networks. (Proulx et al.) [4]

Nodes in a biological network usually represent biological components of interest such as chromosomes, proteins and can even represent species. Edges indicate interaction between nodes such as regulatory interaction, gene flow, social interactions, or infectious contacts. Typically, real biological populations are characterized by properties such as degree heterogeneity, assortative mixing, non-trivial clustering coefficients, and community structure. The presence of large groups of nodes that are highly interconnected among themselves, but loosely connected with other node groups is an intriguing pattern that is also known as assortative community structure.

In empirical networks, these groups, also called modules or communities, often correspond well with experimentally-known functional clusters within the overall system. Thus, community detection, by examining the patterns of interactions among the parts of a biological system, can help identify functional groups automatically, without prior knowledge of the system's processes. (Pratha Sah et al.) [5]

1.3 Objectives

The main challenge of this work is to determine an algorithm for detecting communities in signed, directed graphs. Community detection results can be then interpreted and exploited in the context of intelligent web applications and services.

At the theoretical level, inspired by the algorithm proposed by Venu Satuluri and Sirinivasan Parthasarathy, we propose a variation that targets signed, directed graphs and introduces a similarity score that is based on the concepts of positive and negative co-citation and co-reference.

At the technical level, we provide a full implementation of the proposed algorithm as well as a multitude of randomly generated signed directed graphs on which we verify the correctness and accuracy of our method.

2 Networks - Graphs

2.1 Theoretical Background

Graph theory effectively concerns itself with the study of relationships. Since graphs can emulate a multitude of real world systems, from layouts of cities, to complicated datasets, graph theory amounts to an incredibly useful tool that can define, measure and simplify the constantly changing variables of a dynamic system. The study of graphs can offer solutions to many problems, in various frameworks, concerning different types, matches and utilising different networks and optimization techniques. Graphs can model complex systems by representing procedures and relationships between objects in a wide variety of fields, delivering a lot of valuable applications.

2.1.1 Graphs

A graph G is an ordered triple (V(G),E(G),ψ) consisting of a nonempty set V(G) of vertices, a set E(G), disjoint from V(G), of edges and an incidence function ψ that associates with each edge of G an unordered pair of (not necessarily distinct) vertices of G. If e is an edge and u and v are vertices such that ψ(e)=uv, then e is said to join u and v; the vertices u and v are called the ends of e [6]. Often, graphs are represented diagrammatically as sets of vertices and edges, where vertices are presented as dots and edges are presented as lines connecting the vertices. Graphs are studied extensively in the field of discrete mathematics.

Graphs are so named because they can be represented graphically,and it is this graphical representation that helps us understand many of their properties. Each vertex is indicated by a point and each edge by a line joining the points which represent its ends. There is no unique way of drawing a graph; the relative positions of points representing vertices and lines representing edges have no significance. A diagram of a graph merely depicts the incidence relation holding between its vertices and edges. We shall however often draw a diagram of a graph and refer to it as the graph itself; in the same spirit, we shall call its points vertices and its lines edges. We should note that two edges in a diagram of a graph may intersect at a point that is not a vertex. Those graphs that have a diagram whose edges intersect only at their ends are called planar, since such graphs can be represented in the plane in a simple manner.

The edges of a graph may be directed or undirected and such graphs can be divided into directed graphs and undirected graphs. The word "graph" was first used in this sense by James Joseph Sylvester in 1878 [7].

2.1.2 Vertices and Edges

A vertex or node is the fundamental component of graphs. Graphs are essentially created by vertices. It is important to note that most of the definitions and concepts in graph

theory are suggested by the graphical representation. The ends of an edge are said to be incident with the edge, and vice versa. The two vertices forming an edge are said to be the endpoints of this edge, and the edge is said to be incident to the vertices. Two vertices that are incident with a common edge are adjacent as are two edges which are incident with a common vertex. An edge with identical ends is called a loop and an edge with distinct ends is called a link. With the term endpoints we refer to two vertices that create an edge and in these cases the edge is said to be incident to the vertices.

Undirected graphs consist of unordered groups of vertices and groups of edges, while directed graphs consist of ordered groups of vertices and groups of arcs (directed edges). When graphs are represented in diagrammatic form, vertices appear as circles with labels, while edges appear as lines or arrows, that connect one vertex to another. Vertices are unable to be divided into sub-vertices and are deemed to have no specific features. Occasionally, depending on each particular application, it is possible for vertices to have some type of structure.

If two edges of a graph have a common vertex, then these two edges are considered adjacent. With the term consecutive we refer to edges of a directed graph in the cases where the head of the first one is the tail of the second one. Similarly, we are calling two vertices adjacent if they have a common edge and consecutive if the first one is the head and the second one the tail of an edge.

2.1.3 Properties of graphs

A graph is considered finite if both its vertex and set and edge set are finite. In this book we study only finite graphs and so the term graph always means finite graph. A graph with just one vertex is called trivial and all other graphs non-trivial.

The graph with only one vertex and no edges is called the trivial graph. A graph with only vertices and no edges is known as an edgeless graph. The graph with no vertices and no edges is sometimes called the null graph or empty graph, but the terminology is not consistent and not all mathematicians allow this object.

In general, the vertices of a graph can possibly be distinguished from each other since they are elements of a set. In this case that specific graph is said to be vertex-labelled. Despite that it is considered best practice to consider vertices as indistinguishable. This also applies to edges. As a result of this graphs with labeled edges are considered edge-labelled. Graphs whose vertices or edges have been assigned with labels are called labelled graphs. On the other hand the graphs that are considered unlabelled consist of vertices that cannot be distinguished and edges that also cannot be distinguished.

2.2 Useful Concepts

In this section we will introduce various useful concepts and terms that we will later use to demonstrate the analytical method for detecting communities in signed directed graphs based on the connectivity patterns among nodes.

Node
A synonym for vertex.

Edge
An edge is along with vertices one of the two basic units out of which graphs are constructed. Each edge has two vertices to which it is attached. These vertices are called endpoints. Edges may be directed or undirected; undirected edges are also called lines and directed edges are also called arcs or arrows. In an undirected simple graph, an edge may be represented as the set of its vertices, and in a directed simple graph it may be represented as an ordered pair of its vertices. An edge that connects vertices x and y is sometimes written xy.

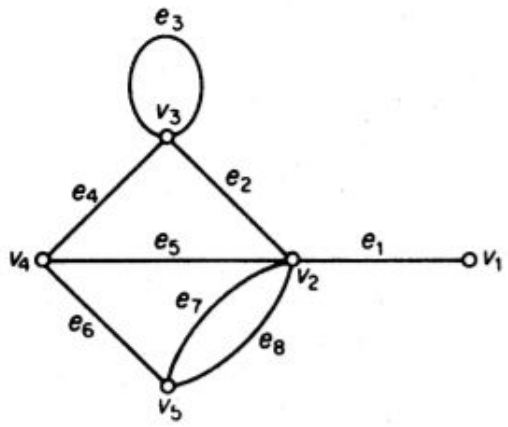
Network
A graph in which attributes (e.g. names,colours etc.) are associated with the nodes and/or edges.

Connectivity
The term connectivity describes the extent to which a graph is connected. It is a concept that is related to the possibility that an edge exists between two random nodes. Connectivity is considered a fundamental notion of graph theory. The connectivity of a graph is an important measure of its resilience as a network.

Degree
The degree d(v) of a vertex v in a graph G, is the number edges of G that are incident to the vertex v, each loop counting as two edges. The degree of a graph G is the maximum of the degrees of its vertices. The degree of a graph G is denoted as $\Delta(G)$ while the minimum degree of graph G (which is the minimum of its vertex degrees) is denoted as $\delta(G)$.
The degree of v in G may be denoted dG(v), d(G), or deg(v). The total degree is the sum of the degrees of all vertices; In any graph the number of vertices of odd degree is even

Loop

An edge with identical ends is called a loop and an edge with distinct ends is called a link. A loop or self-loop is an edge both of whose endpoints are the same vertex. It forms a cycle and as such loops are not allowed in simple graphs.



Adjacency Matrix

The adjacency matrix of a graph is a matrix whose rows and columns are both indexed by vertices of the graph, with a one in the cell for row i and column j when vertices i and j are adjacent, and a zero otherwise.

Multigraph

A multigraph is a graph in which multiple adjacencies and self loops are possible . Any graph that is not necessarily simple can be a multigraph.

Similarity

Similarity is a measure that determines whether two nodes are similar.Two nodes are considered similar if they share many of the same neighbors. Usually node similarity algorithms compare a set of nodes based on the nodes they are connected to.
Similarity in network analysis occurs when two nodes fall in the same equivalence class.

Subgraph

A subgraph of a graph G is another graph formed from a subset of the vertices and edges of G. The vertex subset must include all endpoints of the edge subset, but may also include additional vertices. A spanning subgraph is one that includes all vertices of the graph; an induced subgraph is one that includes all the edges whose endpoints belong to the vertex subset.

Weight
The term weight describes a numerical value, attached to a vertex or an edge of a class. Weight functions as a label.

Weighted graph
If weights have been attached to the vertices or the edges of a graph, this graph is called a weighted graph.

2.3 Directed Graphs

A directed graph is a graph that consists of vertices and edges.The edges of a directed graph have a particular direction attached to them. All graphs whose edges have a direction are directed graphs. A directed graph D is an ordered triple (V(D), A(D), ψ) consisting of a nonempty set V(D), of arcs, and an incidence function ψ that associates with each arc of D an unordered pair of (not necessarily distinct) vertices of D. If a is an arc and u and v are vertices such that ψ(a)=(u,v),then a is said to join u to v; u is the tail of a and u is its head. For convenience, directed graphs are often abbreviated as digraphs. A digraph D′ is a subdigraph of D if $V(D') \subseteq V(D), A(D') \subseteq A(D)$ and ψ is the restriction of ψ to A(D′). The terminology and notation for subdigraphs is similar to that used for subgraphs.

With each digraph D we can associate a graph G on the same vertex set; corresponding to each arc of D there is an edge of G with the same ends. This graph is the underlying graph of D. Conversely, given any graph G we can obtain a digraph from G by specifying, for each link an order on its ends. Such a digraph is called orientation of G.

Just as with graphs, digraphs have a simple pictorial representation. A digraph is represented by a diagram of its underlying graph together with arrows on its edges, each arrow pointing towards the head of the corresponding arc.

Every concept that applies to graphs by default applies to digraphs too. Despite that, the opposite is not necessarily true since there are many different concepts that involve the notion of orientation and they apply only to digraphs. [8]

If we consider an arrow (x,y) , with y being the head and x the tail of the arrow, y is considered a direct successor of x and x is considered a direct predecessor of y is reachable from x. If we subsequently consider an arrow (y,x), that arrows is named the inverted arrow of (x,y)

2.3.1 In- and Out-Degree

The indegree of a vertex is a concept that describes the number of head ends that are adjacent to a vertex. Similarly the outdegree of a vertex describes the number of tail ends adjacent to a vertex. The indegree of a vertex v is denoted $deg_{in}(v)$ and its outdegree is denoted $deg_{out}(v)$.

Concerning a vertex v, if it has $deg_{in}(v) = 0$ it is called a source, since it functions as the origin of every outcoming vertex. Similarly, if a vertex has $deg_{out}(v) = 0$, it is called a target, since it is the end of every incoming vertex. [9] The degree sum formula states that, for a directed graph $\sum_{v \in V} deg\,(v) = \sum_{v \in V} deg^{+}\,(v) = |A|$

2.3.2 Connectivity

A directed graph is weakly connected (or just connected) if the undirected underlying graph obtained by replacing all directed edges of the graph with undirected edges is a connected graph while a directed graph is strongly connected or strong if it contains a directed path from x to y and a directed path from y to x for every pair of vertices {x, y}. The strong components are the maximal strongly connected subgraphs [10]

2.4 Signed Graphs

In a weighted graph, an edge with a positive weight denotes similarity or proximity of its endpoints. For many reasons, it is desirable to allow edges labeled with negative weights, the intuition being that a negative weight indicates dissimilarity or distance. Weighted graphs for which the weight matrix is a symmetric matrix in which negative and positive entries are called signed graphs. A signed graph is essentially a graph in which each edge has a positive or negative sign.

One of the three characteristics of signed graphs that should be studied is whether a signed graph is balanced or not. Signed graphs are considered balanced when the product of edge signs in every cycle is positive. The other two characteristics concern the largest size of a balanced edge set in the graph and what is the smallest number of vertices that should be removed in order for the graph to become balanced.

Signed graphs (with weights (-1, 0, +1)) were introduced as early as 1953 by Frank Harary [13] to model social relations involving disliking, indifference, and liking. At the Center for Group Dynamics at the University of Michigan, Dorwin Cartwright and Harary generalized Fritz Heider's psychological theory of balance in triangles of sentiments to a psychological theory of balance in signed graphs.

Signed graphs where every edge has a direction,in such a way that in positive edges the ends are both directed from one endpoint to the other, and in negative edges either both ends are directed outward to their own vertices or both are directed inward away from the vertices

are deemed to be oriented. An oriented graph is the same as a bidirected graph, and finally, a signed digraph is a directed graph with signed arcs.

Applications

Signed graphs have found extensive applications in social psychology, where social situations are being modelled as signed graphs. People are represented as nodes while positive edges represent friendships and negative edges represent enmities. [13] Signed graphs have also been used in models that study balance theory and even changing international alliances between countries. In natural sciences, and more particularly in physics, signed graphs are used in the general, non-ferromagnetic Ising model, which in turn is used in the study of spin glasses. [14]

Also, using an analytic method initially developed in population biology and ecology, but now used in many scientific disciplines, signed digraphs have found application in reasoning about the behavior of complex causal systems. [15] Such analyses answer questions about feedback at given levels of the system, and about the direction of variable responses given a perturbation to a system at one or more points, variable correlations given such perturbations, the distribution of variance across the system, and the sensitivity or insensitivity of particular variables to system perturbations.

Finally and more importantly, correlation clustering looks for natural clustering of data by similarity. The data points are represented as the vertices of a graph, with a positive edge joining similar items and a negative edge joining dissimilar items. [16]

2.5 Adjacency Matrix

An adjacency matrix is defined as follows:

Let G be a graph with "n" vertices that are assumed to be ordered from v_1 to v_n. The n x n matrix A, in which a_{ij}= 1 if there exists a path from v_i to v_j and a_{ij} = 0 otherwise, is called an adjacency matrix.

The adjacency matrix of a signed graph G on n vertices is an n × n matrix A(G), and each row and each column correspond to a vertex. In row a and column b , the value in the cell c_{ab} is equal to the number of positive edges minus the number of negative edges.

In directed graphs, the adjacency matrix shows an edge from j to i. Given this, the in-degree of a vertex is given by the corresponding row sum and the out-degree is given by the corresponding column sum. The adjacency matrix of a simple labeled graph is the matrix A with A[[i,j]] or 0 according to whether the vertex vj, is adjacent to the vertex vj or not. For simple graphs without self-loops, the adjacency matrix has 0s on the diagonal. For undirected graphs, the adjacency matrix is symmetric. [11]

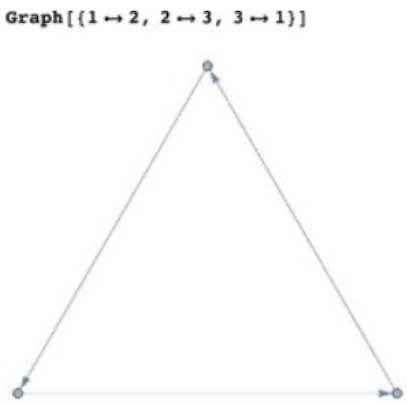


Figure 2: A graph consisting of three nodes and three edges

The adjacency matrix of the shown directed graph is the following:

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

3 Clustering Algorithms

3.1 Background

Clustering algorithms are widely used in order to solve problems where the main task is to group a set of objects in such a way that objects in the same group (called cluster) are more similar to each other than to those in other groups. Clustering is a building block and a main objective of exploratory data mining, statistical data analysis, pattern recognition, image analysis, machine learning and bioinformatics.

Clustering involves specific algorithms that are used for each specific problem to be solved. The algorithms used can be very dissimilar in their definition of a cluster and the procedures used and even the problems that the algorithms seek to solve can differ even more as to the definition of similarity in order to classify the objects of a graph.

The concept of a cluster does not have a universal accurate definition and this is the cause of the existence of various algorithms that are so dissimilar with each other. Different problems demand the use of different algorithms, clustering methods , procedures and models and even sometimes different definitions of concepts, according to each research problem.

Certain concepts as to what is a cluster are more widely used than others. Some of them include classes of vertices whose distance with each other is very small compared to vertices outside the class, while others involve specific statistical distributions. Each specific data set will require a suitable treatment, which can be very dissimilar to the treatment of other datasets, according to the intended result in each case. As a result of this, clustering involves a broad number of techniques, and it is not an automatic function, but rather a procedure that involves understanding and interactive multi-objective optimization that involves trial and error.

Well known and widely used clustering algorithms include K-means, DBScan, Spectral clustering, Hierarchical clustering, principal component analysis and Affinity Propagation. All these algorithms use completely different approaches to clustering.

3.2 Affinity Propagation Clustering

Affinity Propagation is an unsupervised machine learning algorithm that is particularly well suited for problems where we don't know the optimal number of clusters. Clustering data by identifying a subset of representative examples is important for processing signals and detecting patterns in data. Such exemplars can be found by randomly choosing an initial subset of data points and then iteratively refining it, but this works well only if that initial choice is close to a good solution. Affinity propagation takes as input measures of similarity

between pairs of data points and real valued messages are changed between data points until a high quality set of exemplars and corresponding clusters gradually emerges.

A lot of clustering algorithms demand the number of clusters to be determined as input before clustering takes place. This is not a good fit for this project and as such an algorithm that does not require the number of clusters would be preferable. Similar to other clustering algorithms affinity propagation finds exemplars. By exemplars we mean members of the dataset that are representative of clusters. Affinity propagation is based on passing messages between data points. [17] .

We used affinity propagation to employ clustering in our datasets and identify representatives of clusters. In general affinity propagation finds clusters with much lower error than other methods and does so in less time.

The similarity between two points is quantified by a function called S, which is used as input.Once the function is passed, the algorithm uses two different matrices which are updated in every iteration. These matrices are the responsibility matrix and the availability matrix. The former contains values which are used in order to determine how fit a data point is to function as the exemplar of a cluster, while the values of the latter concern how suitable it would be for a data point to pick another as an exemplar.

Affinity Propagation was first published in 2007 by Brendan Frey and Delbert Dueck in Science [18] . In layman's terms, in Affinity Propagation, each data point sends messages to all other points informing its targets of each target's relative attractiveness to the sender. Each target then responds to all senders with a reply informing each sender of its availability to associate with the sender, given the attractiveness of the messages that it has received from all other senders. Senders reply to the targets with messages informing each target of the target's revised relative attractiveness to the sender, given the availability messages it has received from all targets. The message-passing procedure proceeds until a consensus is reached. Once the sender is associated with one of its targets, that target becomes the point's exemplar. All points with the same exemplar are placed in the same cluster [19] .

Supposing a dataset with N nodes, then each node is represented as a data point in a N-dimensional space. Every cell in the similarity matrix is calculated from the differences between nodes, so that the diagonal of the similarity matrix only has zeroes. The algorithm will converge around a small number of clusters if a smaller value is chosen for the diagonal, and vice versa. The algorithm starts off by constructing an availability matrix with all elements set to zero. Then it calculates every cell in the responsibility matrix using the following formula:

$$r(i,k) \;=\; s(i,k) \;-\; max\{a(i,k') + s(i,k')\}$$

where $k' \neq k$, i refers to the row and k refers to the column of the associated matrix.

The algorithm uses a separate equation for updating the elements on the diagonal of the availability matrix than it does the elements off the diagonal of the availability matrix. The proceeding formula is used to fill in the elements on the diagonal:

$$a(k,k) = \sum_{i' \neq k} max\{0, r(i',k)\}$$

where i refers to the row and k the column of the associated matrix.

This equation sums up all positive values along the column except for the row whose value is equal to the column in question. Once all calculations are completed we end up with the availability matrix.

With regard to the criterion matrix, each cell is simply the sum of the availability matrix and responsibility matrix at that location:

$$c(i,k) = r(i,k) + a(i,k)$$

The highest criterion value of each row is designated as the exemplar. Rows that share the same exemplar are in the same cluster. It's worth noting that in this example the values of all 3 matrices range over the same scale. Still, if there are sets of values in different scales they must be normalized prior to training.

3.3 Fine-tuning AP parameters

In the context of this thesis we leveraged the Affinity Propagation implementation of the scikit-learn Python ML framework. This implementation performs data clustering given certain parameters that can be tuned for maximum performance [20]. Here are some parameters that we studied and tuned:

- **damping**: extent to which the current value is maintained relative to incoming values (weighted 1 - damping). This is needed to avoid numerical oscillations when updating these values.
- **max_iter**: the maximum number of iterations. Once the number is reached iterations stop. Default value is set to 200.
- **convergence_iter**: the number of iterations with no change in the number of estimated clusters that stops the convergence.
- **affinity**: determines whether euclidean or precomputed distance is used and as such affinity was set to precomputed, since our main objective in using the algorithm was to take the similarity matrix of the graph as an input.
- **random_state**: is a pseudo-random number generator to control the starting state. An integer can be used for reproducible results across function calls. We set this parameter

to 0 (which is the default number in latest versions of the library) to avoid unnecessary warnings.

- **verbosity**: controls whether the output is verbose or not. We set this parameter for more detailed output and errors.

3.4 K-means Clustering

K-means is a clustering algorithm that was first introduced in signal processing, and whose aim is to divide a number of observations or data points into a number of clusters. The number of observations or data points is often called n, and the number of clusters is called k. The clustering should take place in such a way that every observation is sorted into the cluster with the nearest mean. The mean of every cluster functions as a cluster center and is representative of a cluster. The use of k-means clustering lowers to a minimum the variances of objects that can be found within each cluster, but it does not lower to a minimum the regular euclidean distances. [21] K-means is related to the k-nearest neighbor algorithm, which is a widely used method of machine learning and functions as a classifier.

The term "k-means" was first used by James MacQueen in 1967 [22], although the idea goes back to Hugo Steinhaus in 1956. [23] The algorithm was not published as a journal article until 1982 [24] and in 1965, Edward W. Forgy published essentially the same method, which is why it is sometimes referred to as the Lloyd–Forgy algorithm [25].

K-means' goal is to divide a n number of observations $(x1,\ x2,\ ...,\ xn)$ into k groups, $S = \{S1,\ S2,\ ...,\ Sk\}$ having $k\ (\leq n)$, while seeking to lower to a minimum the within-cluster variance (sum of squares) by finding $argmin \sum_{i=1}^{k} \sum_{x \in S} \left\| x - \mu_i \right\|^2 = arg\ min \sum_{i=1}^{k} \left| S_i \right| Var \cdot S_i$ where μ_i is the mean of data points in cluster S_i. This is equivalent to minimizing the pairwise squared deviations of points in the same cluster: $arg\ min \sum_{i=1}^{k} \frac{1}{2|S_i|} \sum_{x,y \in S} ||x - y||^2$. The equivalence can be deduced from identity $\sum_{x \in S} \left\| x - \mu_i \right\|^2 = \sum_{x \in S} (x - \mu_i)(\mu_i - y)$. Since the total variance remains stable, this is identical to maximising the sum of squared deviations between data points in separate clusters. [26].

Two widely accepted algorithms are Lloyd's algorithm and Elkan's algorithm. The average complexity is given by O(k n T), where n is the number of samples and T is the number of iterations. [27]

The algorithm which is referred to as Lloyd's algorithm, which is mentioned above, is an algorithm whose aim is to find evenly spaced sets of data points in subgroups of Euclidean space, and divides these subset into equally sized cells [28]. Similar to the k-means clustering algorithm it targets the centroid of every set and then re-divides the data set, with regard to which group's centroid is the closest.

The algorithm is usually applied directly to the Euclidean plane, but similar algorithms can be applied to n-dimensional spaces, with n reaching higher values, or to spaces with non-Euclidean metrics. [29]

To gain a deeper insight into the k-means internals of the algorithm we studied and tested the K-Means implementation provided by the scikit-learn Python ML framework. In practice, the k-means algorithm is very fast (one of the fastest clustering algorithms available), but it falls in local minima. That's why it can be useful to restart it several times. If the algorithm stops before fully converging (because of tol or max_iter), labels_ and cluster_centers_ will not be consistent, i.e. the cluster_centers_ will not be the means of the points in each cluster. Also, the estimator will reassign labels_ after the last iteration to make labels_ consistent with the predicted labels on the training set.

Once a set of k means m_1 ,m_2 ,.... ,m_k , is given to the algorithm, it moves forward by using consecutively two different procedures: the assignment procedure, and the update procedure. In the assignment procedure the algorithm attaches every data point to the cluster that has the nearest mean, which means that it divides the data points according to the voronoi diagram generated by the means. The update step which takes place next, involves computing the new means of every cluster. The algorithm then proceeds to the assignment procedure again, and the new update procedure after that. In order for the algorithm to converge to a partitioning, the assignment procedure should no longer change in every new step. Often the algorithm attaches the vertices to the most suitable cluster by distance, usually the euclidean distance, but this is not mandatory. Modifications to the k-means clustering have been suggested in order to use distance metrics other than the euclidean distance. The procedure of assignment is often called "expectation" step and the update procedure "maximization" step.

In the present thesis, since the algorithm needs a specific number of clusters in order to function, we use the number of clusters that is specified by the affinity propagation algorithm used above.

In order to initialize the algorithm, the Forgy or the Random Partition methods are often used. The former picks k data points and uses them as starting cluster means , while the latter uses a random method to attach every data point to a cluster and then starts the update procedure. The Forgy method tends to spread the initial means out, while Random Partition places all of them close to the center of the data set. Different authors and different studies can have completely opposing views as to which method initializes the algorithm in an optimal way. Some suggest that the Random Partition method is better suited for algorithms such as the k-harmonic means but for standard k-means algorithms on the other hand, the Forgy method is better suited. Some studies even suggest that widely used initialization techniques are marked by poor performance [30]

It is interesting to note that the algorithm is not assured to converge and that the results possibly depend on external conditions.

In most cases where the datasets have an underlying clustering arrangement, Lloyd's algorithm's number of iterations is small until it reaches convergence. The algorithm is regarded to be of "linear" complexity [31] [32].

The result of k-means can be seen as the Voronoi cells of the cluster means. Since data is split halfway between cluster means, this can lead to suboptimal splits as can be seen in the "mouse" example. The Gaussian models used by the expectation-maximization algorithm (arguably a generalization of k-means) are more flexible by having both variances and covariances.

It is also important to note how K-means performs, when compared to the similar expectation-maximization algorithm (EM). The EM algorithm is in general regarded as better able to handle clusters whose sizes are subject to change better than k-means. On the other hand EM demands that a lot more parameters are optimized first. Also EM's methodology can create problems especially when the problem concerns clusters that disappear or covariance matrices that are not properly conditioned. Finally K-means bears similarities and is associated with nonparametric Bayesian modeling [33]

4 A similarity measure for signed, directed graphs

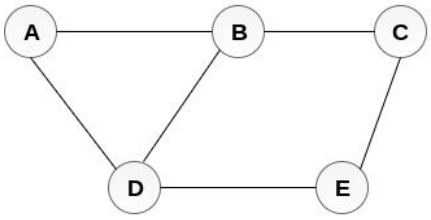
4.1 Adjacency Matrices

With the term adjacency matrix we refer to a squared matrix whose function is to represent a finite graph. The notion of adjacency matrix is rooted in graph theory. Each cell of the adjacency matrix shows whether pairs of vertices are adjacent or not. This is possible with the value of 1 if there is an edge between the two vertices and a 0 if the vertices are not connected. In cases of signed graphs where certain edges have negative value, this is often indicated with the value of -1. The adjacency matrix of a graph, since it is a square matrix, has a size of nxn , where n is the size of the graph.

In general the adjacency matrix of a graph G, is the NxN matrix $A(G) = [a_{ij}]$, in which a_{ij} is the number of edges joining u_i and u_j , where u_i and u_j denote the vertices of the graph.

Concerning o a finite simple graph, its adjacency matrix shall be a square NxN matrix where N denotes the graph size (number of vertices) and the values involve zeroes on the diagonal and the value 1 when vertices are connected with edges. The values of the cells in the diagonal of the adjacency matrix of a simple graph are all zero, given that loops (edges from a vertex that end in itself) are not present. The relationship between a graph and the eigenvalues and eigenvectors of its adjacency matrix is studied in spectral graph theory.

Each edge adds 1 to the appropriate cell in the matrix for undirected graphs , and each loop adds 2. The sum of the values in either the respective row or column in the adjacency matrix allows the degree of the vertex to be observed easily. [35] .

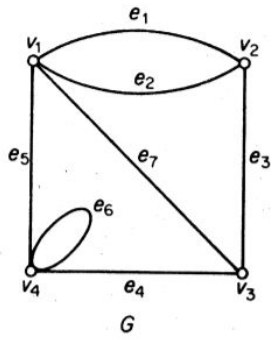


Undirected Graph

	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	0
C	0	1	0	0	1
D	1	1	0	0	1
E	0	0	1	1	0

Adjacency Matrix

It is of crucial importance that we differentiate between the adjacency matrix of a graph and the incidence matrix of a graph. To any graph G there corresponds a vx ε matrix called the incidence matrix of G, which is the matrix $M(G) = [m_{ij}]$, where m_{ij} is the number of times (0,1 or 2) that u_i and ε_j are incident. The incidence matrix of a graph is another different way of specifying the graph. [34] . The adjacency matrix A(G) and the incidence matrix M(G) of a graph are presented below:



G

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
v_1	1	1	0	0	1	0	1
v_2	1	1	1	0	0	0	0
v_3	0	0	1	1	0	0	1
v_4	0	0	0	1	1	2	0

$\mathbf{M}(G)$

	v_1	v_2	v_3	v_4
v_1	0	2	1	1
v_2	2	0	1	0
v_3	1	1	0	1
v_4	1	0	1	1

$\mathbf{A}(G)$

Once we have clearly defined the adjacency matrix, and distinguished this important concept from the incidence matrix , we can move on into defining the matrices A+ and A-, concerning the positive and negative sub-graphs respectively.

A- and A+ matrices are matrices that contain the same elements with the original adjacency matrix of the graph with one key difference: in A- all the positive elements are replaced with zeroes and in A+ all the negative elements are replaced with zeroes. As a result of this procedure A- contains the negative elements of the original adjacency matrix whereas A+ contains just the positive ones.

4.2 Co-citation and Co-reference Matrices

Co-occurrence matrices, such as co-citation, co-word, and co-link matrices, have been used widely in the information sciences. However, confusion and controversy have hindered the proper statistical analysis of this data [39]. Co-occurrence matrices, such as co-citation, co-word, and co-link matrices, provide us with useful data for mapping and understanding the structures in the underlying document sets. Various types of analysis have been carried out on this data and a significant body of literature has been built up, making it an important area of information science (e.g., White & McCain, 1998).

If we assume a directed signed graph $G = (V, E)$, our goal is to cluster nodes in G so that nodes in the same cluster share more common positive and negative in- and out- links than nodes in different clusters. We have defined the adjacency matrix A of said directed graph G and we have also defined the matrices A^+ and A^- as the adjacency matrices of subgraphs with positive and negative links respectively.

The co-reference matrix B (also referred to as bibliographic coupling matrix) was introduced by Kessler [40] in the field of bibliometrics for the sake of counting the number of papers that are commonly cited by two scientific documents, and B[i,j] gives the number of nodes that the nodes i and k both point to in the original directed graph. The co-citation matrix C was introduced by Small [41], again in the field of bibliometrics and C[i,j] gives the number of nodes that commonly point to both i and j in the original directed graph [42].

The next step of the process is to define the matrix $B^+ = A^+(A^+)^T$ that refers to the number of nodes that are commonly cited with positive sign by two nodes, and the matrix $B^- = A^-(A^-)^T$ that denotes the matrix that contains the number of nodes that are commonly cited with negative sign by two nodes. As such $B^+[i,j]$ gives the number of nodes that the nodes i and j both point to with positive sign in the original directed graph and $B^-[i,j]$ gives the number of nodes that the nodes i and j both point to with negative sign in the original directed graph.

Then we define the co-citation matrices for positive and negative subgraphs of G, respectively. They are given by $C^+ = (A^+)^T A^+$ and $C^- = (A^-)^T A^-$ and $C^+[i,j]$ gives the number of nodes that commonly point to both i and j with positive sign in the original directed graph. Similarly $C^-[i,j]$ gives the number of nodes that commonly point to both i and j with negative sign in the original graph.

The similarity between a pair of nodes in a signed directed graph is related to the number of common in- and out- links of nodes. However co-citations are directly related to the in- degrees of the nodes involved and co-references are directly related to the out- degrees of the nodes involved. Given the above, co-reference matrices B^+ and B^+ refer to out-degrees and co-citation matrices C^+ and C^- refer to in-degrees with positive and negative signs, respectively.

It is notable, however, that none of the above mentioned matrices is normalized. When it comes to the actual computation of the similarity matrix to be used as input to the clustering step of our pipeline this could potentially cause problems. Therefore, a normalization factor is absolutely necessary to avoid data anomalies and increase its accuracy. Normalization of the above mentioned matrices was originally thought of as very important, yet we discovered that normalization at the similarity level would be a better fit for our project, since this method produced better results at the validation stage regarding the values of the similarity matrix. Eventually, we decided to normalize similarity scores which means that we calculated the co-reference and co-citation matrices using their non-normalized definitions [43]:

$$B^+ = A^+ \cdot (A^+)^T$$
$$B^- = A^- \cdot (A^-)^T$$
$$C^+ = (A^+)^T \cdot A^+$$
$$C^- = (A^-)^T \cdot A^-$$

Once we calculate the above matrices, we can proceed with the calculation of the incoming and outgoing similarity matrices. To define these similarity matrices we use both positive and negative co-reference and co-citation matrices, using carefully calculated normalization factors.

4.3 Link Balance

In order to determine the similarity between two nodes it would be advisable that we take into account the factor of the balance between positive and negative links. We should avoid the case where two nodes whose sum of links is very large are deemed similar even though they could be sharing just positive or just negative links. Therefore, we should take into account the definition of the link balance between nodes.

Throughout the process we discovered that balancing similarity scores did not yield better results as we originally thought and was eventually ignored in the final stages of the procedure. Yet, since it is one of the possible ways through which this thesis can be extended in the future, we decided to document how we defined and tested link balance since it will be useful for next iterations.

Taking into account the link balance between any two nodes of the original graph is very important but at the same time it is equally important to not allow the balance between negative and positive links to play a big role when determining the similarity between two nodes. In order to take into account the link balance but not allowing it to determine the value of similarity between two nodes we sum both the denominator and the numerator of the fraction with the value of 1. As a result the value of the fraction is close to 1. Finally, since we can choose either negative links to positive links or the opposite, it would be preferable to choose the value that is the smallest. Given the criteria mentioned above, we define positive and negative link balance as:

$$balance_{in}(i,j) = min\left\{\frac{1+B_n^+[i,j]}{1+B_n^-[i,j]}, \frac{1+B_n^-[i,j]}{1+B_n^+[i,j]}\right\}$$

$$balance_{out}(i,j) = min\left\{\frac{1+C_n^+[i,j]}{1+C_n^-[i,j]}, \frac{1+C_n^-[i,j]}{1+C_n^+[i,j]}\right\}$$

Balance of in-links is related to the B^+, B^- co-reference matrices since they describe the in-degrees and balance of out-links is related to the C^+ C^- co-citation matrices since they describe the out-degrees. Once the balance-in and balance-out matrices are computed they can be used as regulatory factors in the calculation of the overall similarity score between any two nodes of the original graph.

4.4 Similarity Score

The term similarity score or similarity measure, refers to a function that evaluates the similarity between two objects. Despite the inexistence of a single, comprehensive, all-encompassing definition, the measures of similarity usually involve, in some sense, the inverse of distance measures: their values are large for similar objects and small for dissimilar ones.

Similarity or distance measures are core components used by distance-based clustering algorithms to cluster similar data points into the same clusters, while dissimilar or distant data points are placed into different clusters. The performance of similarity measures is mostly addressed in two or three-dimensional spaces, beyond which, to the best of our knowledge, there is no empirical study that has revealed the behavior of similarity measures when dealing with high-dimensional datasets.

Concerning spectral clustering the similarity measure is used in order to cluster dataset points, and often to simplify complexities that have to do with the shape of the distribution of data [44] [45] . In the next section we discuss the calculation of similarity matrices.

4.4.1 Similarity Matrices Calculation

Regarding the semantics and calculation of the proposed Similarity Matrix, we shall define two different similarity matrices. One is related to the in-links and shall be called in-similarity , while the other is related to the out-links and shall be called out-similarity. Both of these matrices are essentially a sum and multiplication of matrices that have already been calculated.

Incoming similarity is the sum of B^+, B^- co-reference matrices that involve the in-degrees of nodes. Similarly, outgoing similarity is the sum of C^+, C^- co-citation matrices that involve the out-degrees of nodes.

As discussed above, and after careful consideration regarding the correctness of our algorithm, we decided to normalize the overall similarity matrix before we pass it as input to the clustering algorithm. That is, in-similarity and out-similarity are frequency matrices: the former describes the number of common edges between nodes i and j concerning incoming links, while the latter describes the number of common edges between nodes i and j concerning outgoing links. The two similarity matrices are defined as:

$$S_{in}(i,j) = (B^+[i,j] + B^-[i,j])$$
$$S_{out}(i,j) = (C^+[i,j] + C^-[i,j])$$

Finally, we define the overall similarity matrix as the normalized sum of in- and out-similarity matrices. For each pair of nodes i, j we define the normalization factor as the inverse maximum of the node degrees, i.e., D(i) and D(j) which denote the total sum of edges for nodes i and j respectively.

$$Similarity[i,j] = \frac{S_{in}(i,j) + S_{out}(i,j)}{max(D(i), D(j))}$$

Since , similarity-in and similarity-out denote the number of common edges between nodes i and j concerning in-coming and out-going links respectively, their sum is the total number of

common edges the two nodes share. If the nodes share all their edges, that number is equal to the degree D(i) and the degree D(j), and similarity reaches its maximum value which is 1. This value indicates that the two nodes have 100% similarity.

4.4.2 Calculation of Communities

The problem of detecting communities in graphs is a projection of data clustering where the network's topological properties are only considered for measuring similarities among nodes. Among the existing community detection approaches, the affinity propagation (AP)-based method has been showing promising results and does not require any predefined information such as the number of clusters (communities).

Using the similarity matrix as input to a clustering algorithm, like Affinity Propagation, we are able to divide the original graphs into groups of nodes that follow similar connectivity patterns, i.e., point to and are pointed from similar sets of nodes. Since scikit-learn provides a multitude of well-implemented clustering algorithms we decided to test and use scikit-learn's implementation of the Affinity Propagation algorithm, which is widely adopted and considered efficient and reliable. Our approach is to use the overall similarity matrix discussed in the previous section as precomputed affinity instead of the default euclidean. In turn, Affinity Propagation will return communities of nodes that follow similar connectivity patterns in the original graph.

5 Experimental Evaluation

5.1 Development Environment

In this section we shortly describe the environment and set of tools that we used to implement the proposed algorithm for detecting communities in signed directed graphs. In a nutshell, we chose Python 3.8 for the implementation and more specifically numpy for the calculations between matrices, sklearn for the clustering algorithms and networkx to model graphs.

NetworkX

NetworkX is a Python library for the creation, manipulation and study of the structure, dynamics and functions of complex networks. It provides a multitude of useful features, such as: data structures for graphs, digraphs and multigraphs, standard graph algorithms, models for representing networks, analysis tools and generators for classic, random, and synthetic graphs. Edges can hold arbitrary data while nodes can be either text, images or XML records and it is well tested with over 90% code average.

In general, it is appropriate for use on large graphs that are suitable for real world uses. Graphs whose size is more than 10.000.000 nodes and 100.000.000 edges can be manipulated using networkx. [46] .

Matplotlib

Matplotlib is a python library designed specifically for creating two-dimensional plots of arrays in Python. Its origins are in imitating MATLAB graphics, but it is independent of MATLAB and it is possible to be used in object oriented programming since it provides an object oriented API for embedding plots into applications using general purpose GUI toolkits. Matplotlib is written in Python and uses its numerical and mathematics extension, Numpy, extensively, in order to offer great performance for big arrays.

Matplotlib was created with the mindset that one should be able to make plots with very few commands. As a result, Matplotlib is used extensively for data analysis and data visualization since it can easily create plots. It also provides a procedural pulab interface, created in order to imitate that of MATLAB. Matplotlib was originally written by John D. Hunter, and since then it has an active development community.

The pictures below depict how Matplotlib visualizes some of the randomly generated signed directed graphs for our experiments. The red edges denote negative connections, while the green ones denote positive connections [46] .

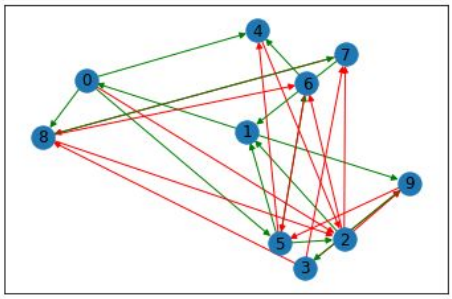


Figure 3: 10 nodes, 30% connected

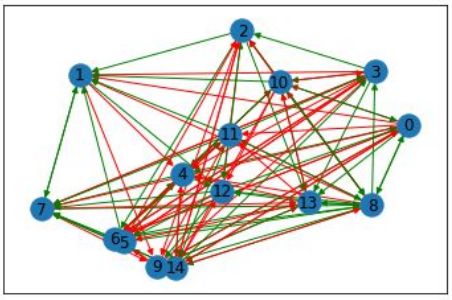


Figure 4: 15 nodes, 50% connected

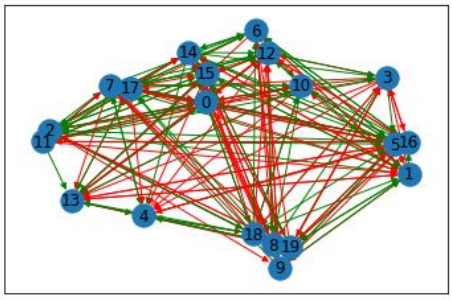


Figure 5: 20 nodes, 30% connected

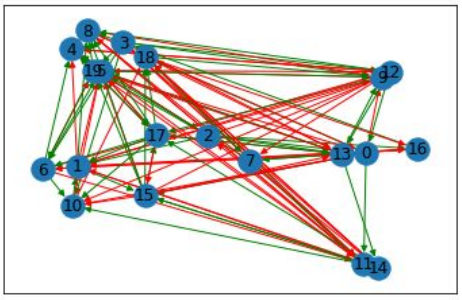


Figure 6: 20 nodes, 50% connected

Figure 3 depicts a graph with 10 nodes and 30% connectivity while figure 4 on the right depicts a graph with 15 nodes and 50% connectivity. Figure 5 and 6 depict a graph with 20 nodes and 30% connectivity and a graph with 20 nodes and 50% connectivity respectively.

Numpy

NumPy stands for Numerical Python. Numpy is a Python library, whose aims include numerical computations, high level mathematical functions and manipulation of large multi-dimensional arrays and matrices. NumPy is mainly used for working with arrays. Its functions can be very useful for linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In general, Python lists serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray. Arrays are very frequently used in data science, where speed and resources are very important. NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This is the main reason why NumPy is faster than lists. Widely used computer vision library OpenCV makes use of Numpy arrays and given that images with multiple channels can

be presented as 3D arrays, indexing, slicing and masking with other arrays are useful methods to manipulate specific pixels of an image

NumPy offers multidimensional arrays, functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops. [49] .

Scikit-learn

Scikit-learn is a widely used machine learning Python library. Its features include classification ,regression and clustering algorithms, as well as tools for dimensionality reduction, model selection and preprocessing. Algorithms include support vector machines, random forests , gradient boosting and others. The first public release was published in 2010.

The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data. Some popular groups of models provided by scikit-learn include:

- **Clustering**: for grouping unlabeled data such as KMeans.
- **Cross Validation**: for estimating the performance of supervised models on unseen data.
- **Datasets**: for test datasets and for generating datasets with specific properties for investigating model behavior.
- **Dimensionality Reduction**: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
- **Ensemble methods**: for combining the predictions of multiple supervised models.
- **Feature extraction**: for defining attributes in image and text data.
- **Feature selection**: for identifying meaningful attributes from which to create supervised models.
- **Parameter Tuning**: for getting the most out of supervised models.
- **Manifold Learning**: For summarizing and depicting complex multi-dimensional data.
- **Supervised Models**: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

Scikit-learn is largely written in Python, and uses numpy extensively for high-performance linear algebra and array operations [48] Scikit-learn is suitable for use with other Python libraries such as matplotlib, numpy, pandas and more.

5.2 Dataset Generator

In order to test our algorithm the first step was to try a real world dataset from Stanford Large Network Dataset Collection. The chosen file was Wiki-RfA, a directed signed graph, which describes wikipedia requests for adminship. For a Wikipedia editor to become an administrator, a request for adminship (RfA) must be submitted, either by the candidate or by another community member. Subsequently, any Wikipedia member may cast a supporting, neutral, or opposing vote.

All votes were parsed since the adoption of the RfA process in 2003 through May 2013. The dataset contains 11,381 users (voters and candidates) forming 189,004 distinct voter/votee pairs, for a total of 198,275 votes (this is larger than the number of distinct voter/votee pairs because, if the same user ran for election several times, the same voter/votee pair may contribute several votes).

This induces a directed, signed network in which nodes represent Wikipedia members and edges represent votes. In this sense, the present dataset is a more recent version of the Wikipedia administratorship election data. Votes can be either positive, negative or neutral. The dataset contains around 10.000 nodes representing the users and 150.000 edges that represent the votes.

When running the algorithm on the dataset, it converged up to a certain number of nodes (around 200) and if the whole file was used the algorithm could not determine a specific number of clusters.
It was hypothesized that since node degrees are used in the calculations of the matrices , graphs with a limited number of edges per node could potentially fill the matrices with zeros and null values, thus rendering the convergence of the algorithm impossible.

As such, a networkx generator that produces random, directed, signed graphs proved to be a useful tool, especially since it is possible to control the number of edges per node, through connectivity, a graph attribute that describes the possibility that an edge exists between two random nodes. The `gnp_random_graph()` function returns a random graph, also known as Erdos-Renyi model graph or binomial graph, with the option of directed edges and positive/negative weights. The model chooses each of the possible edges with a pre-determined probability (referring to the connectivity attribute of the graph) and weights were chosen randomly with positive weight representing a positive sign (visualized as a green line) while negative weight representing a negative sign (visualized as a red line).

The random graph generator produces modular random graphs using only a small number of intuitive and interpretable parameters such as the size of the graph and connectivity of the graph given as a percentage. It produces different graphs for different degree distributions and so it is extremely useful for the interpretation of results by allowing us to study the relationships between convergence , number of nodes , number of edges , connectivity and whether the algorithm converges or not.

Erdos-Renyi model

The Erdos-Renyi *G(n, p)* model was described by Edgar Gilbert for the first time in a 1959 paper that concerned the connectivity threshold of graphs. The term Erdos-Renyi model refers to two models that bear a great deal of resemblance to each other, and are widely used in the generation of random graphs or the study of how a random network evolves. The name Erdos-Renyi comes from the mathematicians Paul Erdos and Alfred Renyi, who presented the models in 1959. Edgar Gilbert subsequently presented the other model independently of Erdos and Renyi [50] [51].

All graphs that consist of a standard set of vertices and a standard number of edges are equivalently possible in the model of Erdos and Renyi. Conversely, in the model created by Gilbert every edge adheres to a standard probability of presence or absence, that is independent of the rest of edges. It is possible for probabilistic methods to prove the existence of graphs that meet different sets of properties , or to offer defining characteristics of the meaning of each property for every graph.. The two variants of the Erdos-Renyi random graph model that are also very similar to each other, are presented here:

First Variant: Erdos-Renyi

The G(n, M) model , presented by Erdos and Renyi, a random choice of a graph takes place. The graph is picked from a group of graphs that have a number of vertices n and a number of edges M.

Second Variant: Gilbert

In the *G*(*n*, *p*) model, presented by Gilbert, and which is used in the present thesis, the construction of a graph takes place and the connection of nodes happens randomly. Every edge gets incorporated in the graph with a probability p, which is completely unconstrained from the rest of the edges. As such, the rest of graphs with a number of nodes n , and a number of edges M, have equal probability of $p^M(1-p)^{\frac{n}{2}-M}$.

In the model presented by Gilbert, the parameter p acts as a function of weight and every increase from 0 to 1, the chance that the model will encompass graphs with a greater number of edges a rises, and the chance that the model will encompass graphs with a smaller number of edges lowers.

A graph in G(n, p) has on average $\frac{n}{2}p$ edges. An interesting observation is that the distribution of the degree of any particular vertex is binomial [52] It should also be note that:

$P(deg(v)=k) = (\frac{n-1}{k})p^k(1-p)^{(n-1)/k}$, where n is the total number of vertices in the graph and since: $P(deg(v)=k) \rightarrow (\frac{(np)^k e^{-np}}{k!})$ $as\ n \rightarrow \infty\ and\ np = constant$ the distribution follows the Poisson distribution for large n and np = const.

5.3 Results

In order to verify the correctness of the implementation and to check the way the algorithm works at large scale datasets, different combinations of graph sizes and different values of node connectivity for each graph were tested.

The metrics that were deemed most important were the number of clusters , whether the scaling algorithm converged or not and total elapsed time. Connectivity is closely connected to the number of edges and the number of nodes and is presented at the tables below along with other metrics. Results are presented in each table for different combinations of number of nodes and connectivity/number of edges. Convergence is described with a yes or no, depending on whether the algorithm converged or not, and in cases where the algorithm converged, but with arbitrary number of clusters, it is denoted at the no. of clusters column.This happens when all samples have mutually equal similarity scores. As stated above at the dataset generator section, the number of edges and corresponding connections between nodes are determined randomly, and as such all nodes have equal similarity scores. This is a special case in which the affinity propagation algorithm reaches its limits, similarity scores between nodes are identical, and the algorithm groups the nodes of the graph in an arbitrary number of clusters. Finally, different connectivity percentages were chosen indicatively for each graph size, in order to determine the relationship between connectivity and the detection of communities in the original graph.

We present the results grouped by the size of the network for different numbers of edges and connections.

No. of nodes	Connectivity	No. of edges	No. of clusters	Convergence	Elapsed Time
5	5.0%	4	5	arbitrary	0.015
5	10.0%	5	2	yes	0.008
5	25.0%	6	2	yes	0.009
5	50.0%	13	3	yes	0.004
5	75.0%	13	2	yes	0.004

Table 1: This table shows the results for a graph of 5 nodes and connectivity scaling from 1.00% up to 75.0%

No. of nodes	Connectivity	No. of edges	No. of clusters	Convergence	Elapsed Time
10	5.0%	13	3	yes	0.010
10	10.0%	12	4	yes	0.009
10	25.0%	20	3	yes	0.008
10	50.0%	45	0	no	0.025
10	75.0%	60	2	yes	0.004

Table 2: This table shows the results for a graph of 10 nodes and connectivity scaling from 1.00% up to 75.0%

No. of nodes	Connectivity	No. of edges	No. of clusters	Convergence	Elapsed Time
25	5.0%	30	0	no	0.025
25	10.0%	66	6	yes	0.007
25	25.0%	145	0	no	0.028
25	50.0%	304	7	no	0.010
25	75.0%	444	0	no	0.028

Table 3: This table shows the results for a graph of 25 nodes and connectivity scaling from 1.00% up to 75.0%

No. of nodes	Connectivity	No. of edges	No. of clusters	Convergence	Elapsed Time
50	5.0%	112	12	yes	0.014
50	10.0%	225	10	yes	0.027
50	25.0%	587	10	yes	0.020
50	50.0%	1242	11	yes	0.026
50	75.0%	1828	0	no	0.056

Table 4: This table shows the results for a graph of 50 nodes and connectivity scaling from 1.00% up to 75.0%

No. of nodes	Connectivity	No. of edges	No. of clusters	Convergence	Elapsed Time
100	5.0%	485	20	yes	0.069
100	10.0%	983	19	yes	0.081
100	25.0%	2440	18	yes	0.125
100	50.0%	4957	16	yes	0.176
100	75.0%	7470	17	yes	0.234

Table 5: This table shows the results for a graph of 100 nodes and connectivity scaling from 1.00% up to 75.0%

No. of nodes	Connectivity	No. of edges	No. of clusters	Convergence	Elapsed Time
250	5.0%	3126	39	yes	0.435
250	10.0%	6282	40	yes	0.668
250	25.0%	15570	34	yes	0.210
250	50.0%	31211	36	yes	2.620
250	75.0%	46680	34	yes	3.447

Table 6: This table shows the results for a graph of 250 nodes and connectivity scaling from 1.00% up to 75.0%

No. of nodes	Connectivity	No. of edges	No. of clusters	Convergence	Elapsed Time
500	5.0%	12366	73	yes	3.109
500	10.0%	24811	74	yes	4.096
500	25.0%	62130	62	yes	10.507
500	50.0%	124814	59	yes	24.326
500	75.0%	187096	63	yes	22.468

Table 7: This table shows the results for a graph of 500 nodes and connectivity scaling from 1.00% up to 75.0%

No. of nodes	Connectivity	No. of edges	No. of clusters	Convergence	Elapsed Time
1000	5.0%	50034	135	yes	17.423
1000	10.0%	99996	123	yes	30.996
1000	25.0%	250741	113	yes	73.885
1000	50.0%	498861	106	yes	152.308
1000	75.0%	749974	109	yes	181.101

Table 8: This table shows the results for a graph of 1000 nodes and connectivity scaling from 1.00% up to 75.0%

No. of nodes	Connectivity	No. of edges	No. of clusters	Convergence	Elapsed Time
2500	5.0%	311768	0	no	198.125
2500	10.0%	623571	0	no	409.011
2500	25.0%	1561236	0	no	978.840
2500	50.0%	3124396	242	yes	1835.197
2500	75.0%	4686305	234	yes	2766.257

Table 9: This table shows the results for a graph of 2500 nodes and connectivity scaling from 1.00% up to 75.0%

No. of nodes	Connectivity	No. of edges	No. of clusters	Convergence	Elapsed Time
5000	5.0%	1249425	511	yes	2576.002
5000	10.0%	2497599	0	no	5017.518
5000	25.0%	6247370	453	no	11623.403
5000	50.0%	9999284	0	no	22951.815
5000	75.0%	18745660	0	no	56009.510

Table 10: This table shows the results for a graph of 5000 nodes and connectivity scaling from 1.00% up to 75.0%

6 Discussion

6.1 Similarity and Clustering Validation

To verify the correctness of the proposed algorithm we had to check the extent to which the similarity scores for individual pairs of nodes indeed reflected patterns in the original graph. In this respect, we have defined similarity as a percentage and we use the definition of similarity measure described above, dividing the values of the sum of in-similarity and out-similarity (which indicates the number of common edges) with the value of total possible common edges. This division is a type of normalization and it means that the values of our final matrix consists of percentages. Every value in the original similarity matrix is divided by the number of possible edges that two nodes can have in common, and is the maximum between the number of edges of node i and the number of edges of node j. Since normalization takes place later in the process, this time we ignore balance and the normalization factors at the level of matrices B and C, since we normalize the matrix at the end

We demonstrate a relatively small cluster of 7 nodes with the aim to examine the values of the similarity matrix. The relationships between nodes were such that the clusters we expected were obvious.

In randomly generated, large, dense graphs, it is quite common that relationships between nodes are very complex. This makes it really hard to verify whether the clustering of the nodes is meaningful since connectivity patterns are not always easy to detect and quantize. Given the aforementioned theoretical background, we expect the clustering algorithm to put the nodes that are commonly cited by another node, or two other nodes, in the same cluster, and the nodes that commonly point to another node, or a set of nodes, in the same cluster respectively.

In order to define the new measure of similarity as percentage we will use the equations that are listed below, and have been explained in depth in previous sections.

$$B^{+} = A^{+} \cdot (A^{+})^{T}$$
$$B^{-} = A^{-} \cdot (A^{-})^{T}$$
$$C^{+} = (A^{+})^{T} \cdot A^{+}$$
$$C^{-} = (A^{-})^{T} \cdot A^{-}$$

As mentioned above, $B^{+}[i,j]$ gives the number of nodes that the nodes i and j both point to with positive sign in the original directed graph and $B^{-}[i,j]$ gives the number of nodes that the nodes i and j both point to with negative sign in the original directed graph.

On the other hand, $C^{+}[i,j]$ gives the number of nodes that commonly point to both i and j with positive sign in the original directed graph. Similarly $C^{-}[i,j]$ gives the number of nodes that commonly point to both i and j with negative sign in the original graph.

Concerning similarity measures, we will first define the matrices Sin and Sout. Here Sin(i,j) and Sout(i,j) matrices denote the number of common edges that the pair of nodes i and j share, with Sin referring to incoming edges and Sout referring to outcoming ones. Dividing the sum of Sin and Sout with the maximum number of edges we have a percentage that denotes similarity.

$$S_{in}(i,j) = (B^{+}[i,j] + B^{-}[i,j])$$
$$S_{out}(i,j) = (C^{+}[i,j] + C^{-}[i,j])$$

$$Similarity[i,j] = \frac{S_{in}(i,j) + S_{out}(i,j)}{max(D(i), D(j))}$$

We expect this measure of similarity to have the value of 1 in the diagonal, and the same value in pairs of nodes that share the same edges. Accordingly, pairs of nodes that do not share common edges would have the value of zero.

In order to test the results we conducted an experiment using graphs with a small number of nodes. The 7-node graph that is depicted below is simple enough to be understood at a glance. The clusters that we expect should also be obvious.

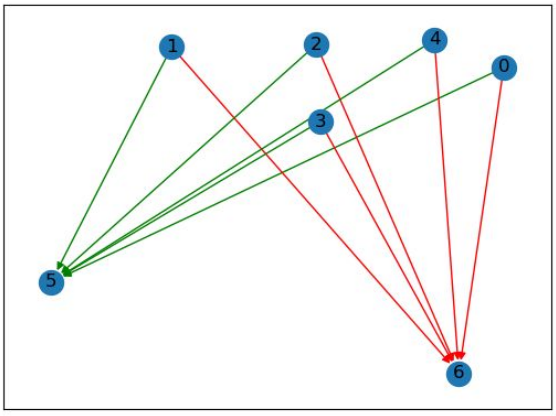


Figure7 : A seven node graph created with 2 communities

```
g = nx.DiGraph()

g.add_edge(0, 5, weight=1)
g.add_edge(1, 5, weight=1)
g.add_edge(2, 5, weight=1)
g.add_edge(3, 5, weight=1)
g.add_edge(4, 5, weight=1)

g.add_edge(0, 6, weight=-1)
g.add_edge(1, 6, weight=-1)
g.add_edge(2, 6, weight=-1)
g.add_edge(3, 6, weight=-1)
g.add_edge(4, 6, weight=-1)

for i, (u, v, d) in enumerate(g.edges(data=True)):
    d["color"] = "green" if d["weight"] > 0 else "red"
    # print(u, v, d)

return g
```

Figure 8: Commands for graph in figure 7

Since nodes 0, 1, 2, 3 and 4 refer to node 5 with positive sign and are cited by node 6 with negative sign, we expect the nodes 0, 1, 2, 3 and 4 to be in the same cluster, while node 5 comprises a cluster of its own, as does node 6. The similarity matrix is depicted below:

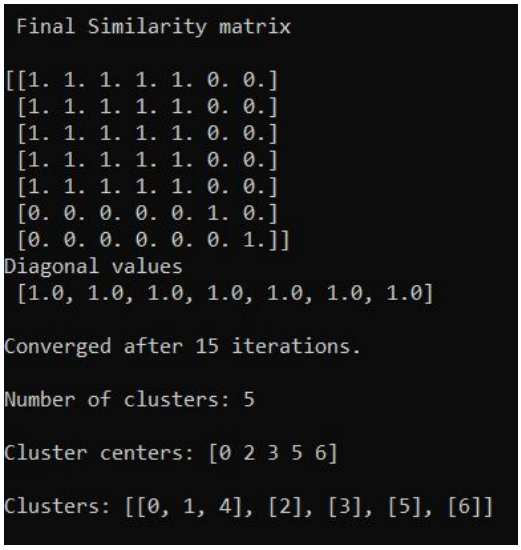


Figure 9: Similarity Matrix for graph in figure 6

The similarity matrix for the graph in figure 7, along with the clustering that resulted from the algorithm shown in figure 9. The values in the similarity matrix were as expected, since the nodes 0-4 share the same edges and are thus 100% similar, while node 5 shares 100% similarity with itself and 0% with the other nodes, and the same applies to node 6. As such the expected values involved the value of 1 in every cell except for lines 5 and 6 and columns 5 and 6 which should have zeroes. Also the cells (5,5) and (6,6) should also have the value of 1 since nodes 5 and 6 share absolute similarity with themselves. The clusters that were returned by the algorithm differed from the clusters that we expected. Instead of clustering nodes 0, 1, 2, 3 and 4 together in the same cluster, the algorithm put node 2 a cluster of its own, and node 3 on its own as well. On the other hand, nodes 5 and 6 were clustered correctly.

We also created a second, larger graph in order to test the results further, this time with 14 nodes, which is depicted below. The clusters that we expect should also be obvious.

Since nodes 0, 1, 2, 3 and 4 refer to node 5 with positive sign and are cited by node 6 with negative sign, we expect the nodes 0, 1, 2, 3 and 4 to be in the same cluster, while node 5 comprises a cluster of its own, as does node 6. The similarity matrix is depicted below:

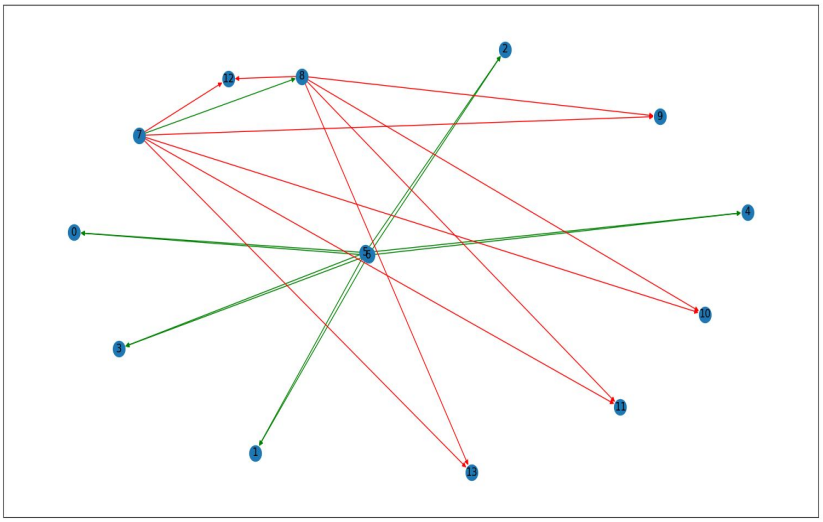


Figure 9: A fourteen node graph, created to validate algorithm results

```python
def generate_directed_signed_graph_g1():
    """Generate hardcoded graph 1."""
    g = nx.DiGraph()

    g.add_edge(5, 0, weight=1)
    g.add_edge(5, 1, weight=1)
    g.add_edge(5, 2, weight=1)
    g.add_edge(5, 3, weight=1)
    g.add_edge(5, 4, weight=1)

    g.add_edge(6, 0, weight=1)
    g.add_edge(6, 1, weight=1)
    g.add_edge(6, 2, weight=1)
    g.add_edge(6, 3, weight=1)
    g.add_edge(6, 4, weight=1)

    g.add_edge(5, 6, weight=1)
    g.add_edge(7, 8, weight=1)

    g.add_edge(7, 9, weight=-1)
    g.add_edge(7, 10, weight=-1)
    g.add_edge(7, 11, weight=-1)
    g.add_edge(7,12,weight=-1)
    g.add_edge(7,13,weight=-1)

    g.add_edge(8, 9, weight=-1)
    g.add_edge(8, 10, weight=-1)
    g.add_edge(8, 11, weight=-1)
    g.add_edge(8,12,weight=-1)
    g.add_edge(8,13,weight=-1)

    for i, (u, v, d) in enumerate(g.edges(data=True)):
        d["color"] = "green" if d["weight"] > 0 else "red"

    return g
```

Figure 10: The edges of the second graph

Given the edges of the second graph, we expected the nodes 0,1,2,3 and 4 grouped together since they are positively cited by node 5 and negatively by node 6. Node 5 and 6 should each belong to a single cluster on their own, since they do not share any common edges with any other node. Nodes 9, 10, 11, 12 and 13 should belong to the same cluster since they are positively by node 8 and positively by node 7. Node 7 should belong to a single cluster on its own. The same applies for node 8. Results are shown below:

```
Final Similarity matrix

[[1.         1.         1.         1.         1.         0.         0.16666667 0.         0.         0.         0.         0.         0.         0.        ]
 [1.         1.         1.         1.         1.         0.         0.16666667 0.         0.         0.         0.         0.         0.         0.        ]
 [1.         1.         1.         1.         1.         0.         0.16666667 0.         0.         0.         0.         0.         0.         0.        ]
 [1.         1.         1.         1.         1.         0.         0.16666667 0.         0.         0.         0.         0.         0.         0.        ]
 [1.         1.         1.         1.         1.         0.         0.16666667 0.         0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         1.         0.83333333 0.         0.         0.         0.         0.         0.         0.        ]
 [0.16666667 0.16666667 0.16666667 0.16666667 0.16666667 0.83333333 1.         0.         0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.         0.         1.         0.83333333 0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.83333333 1.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         1.         1.         1.         1.         1.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         1.         1.         1.         1.         1.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         1.         1.         1.         1.         1.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         1.         1.         1.         1.         1.        ]
 [0.         0.         0.         0.         0.         0.         0.         0.         0.         1.         1.         1.         1.         1.        ]]
Diagonal values
 [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

Converged after 60 iterations.

Number of clusters: 4

Cluster centers: [0 5 7 9]

Clusters: [[0, 1, 2, 3, 4], [5, 6], [7, 8], [9, 10, 11, 12, 13]]
```

Figure 11: Similarity matrix for graph in fiureg 19. Note that values in the diagonal are equal to 1.

It is interesting to note that the values in the diagonal of the similarity are equal to 1. The values in the diagonal denote how similar a node is with itself. As such the value of 1 denotes 100% or full similarity.

6.2 Conclusion

The most important verification step of our process was to check for the values in the diagonal of the similarity matrix. Since the value of 1 appears in all our tests, it is obvious that every node shares 100% similarity with itself. Also, the validation process proved that the algorithm can find clusters correctly in a small graph of 15 nodes. It is important to note that the algorithm works in graphs of up to 5.000 nodes and produces tangible results.

Judging from the above, it is very important that the normalisation takes place at the similarity level, since this important part makes sure that zeroes do not appear at the value of any denominator. This also makes sure that the values in the diagonal of the similarity matrix are equal to 1, since the denominator contains the sum total of all the edges the two nodes could have in common, and the numerator contains the number of common edges that the two nodes share. In the diagonal of the similarity matrix these two values are equal.

The balance factor, though eventually abandoned, could potentially be in the future, in special cases of graphs where the appearance of pairs of nodes with many common positive edges but no negative common ones, makes clustering difficult.

At this point we should also note that arbitrary clustering took place extensively at the first step of the process, where normalization took place in matrices B and C, and not at the similarity level. This issue was fixed once normalisation happened at the level of similarity, meaning that similarity in and similarity out matrices were computed using just the B and C matrices, directly calculated from their respective definitions. Then, the total similarity score was calculated by adding incoming and outgoing similarity matrices and dividing by the number of possible common edges the two nodes could share (max(D(i), D(j)).

Viewing similarity as a percentage was crucial to the process. Incoming and outgoing similarity were used as measures of incoming and outgoing common edges respectively, and were essential in the final calculation of the similarity matrix. The arbitrary clustering that occured after these steps was due to the graph having an especially low number of edges (in cases where low connectivity (<5%) was combined with a very low number of nodes (e.g. 5)). Of course we should always take into account the fact that the data was synthetic and that the graphs created using the custom-made dataset generator were random, meaning that clusters were not always present, especially in these special cases of small graphs with low connectivity.

In general, the algorithm is capable of calculating adequately the similarity matrices it was intended to. These matrices can effectively be used to cluster datasets and results are promising for the next attempts.

6.3 Future Research

Given the fact that the single most important measure for clustering is the diagonal of the similarity matrix, the similarity measure that is proposed here functions adequately. The value of the cells in the diagonal of the similarity matrix, showed exceptional consistency in

taking the value of 1, just as it should. This means that every node in the graph shares a 100% similarity with itself.

Also, it is important to note that clustering showed quite promising results. One of the main goals for any future research project could be to run our algorithm and implementation on large scale datasets of signed, directed graphs for which communities are pre-determined. This would make large-scale validation plausible and, in turn, help one define the accuracy of the proposed clustering algorithm in the real-world.

In this context, the algorithm could be improved by integrating the regulatory factor called balance, the use of which has been outlined above. In cases where imbalances between positive and negative edges between two nodes of a graph exist, the notion of balance could be of crucial importance in determining the clusters that the two nodes belong to. In future attempts, the integration of the balance factor could be very important.

On a different note, more research is needed in order to verify the performance of the Affinity Propagation algorithm. An interesting idea would be to compare the results of the Affinity Propagation algorithm with those of a different clustering algorithm, such as k-means or k-medoids. Alternatively, we could use two different clustering algorithms in the same pipeline: the first clustering algorithm could calculate a rough number of clusters for the given graph, while the second one could use the output of the first one as input to reach more precise results. This would allow us to combine different approaches, approximate the number of clusters before knowing the communities and evaluate our algorithm and implementation from a different angle.

7 Source Code

The implementation of the proposed algorithm is open-source and available at the following GitHub repository:

https://github.com/manosandroulidakis/msc

8 Bibliography

[1] Zhou, Sheng, et al. "HAHE: Hierarchical Attentive Heterogeneous Information Network Embedding." https://arxiv.org/abs/1902.01475
[2] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, Andrew S. Tomkins
The web as Graph: Measurements , Models and Methods, Publisher Name: Springer, Berlin, Heidelberg , doi : https://doi.org/10.1007/3-540-48686-0_1
[3] David F. Nettleton, Data mining of social networks represented as graphs
doi: https://doi.org/10.1016/j.cosrev.2012.12.001
[4] Stephen R. Proulx, Daniel E.L. Promislow, Patrick C. Phillips, Network thinking in ecology and evolution, Trends in Ecology & Evolution, Volume 20, Issue 6, 2005, Pages 345-353, ISSN 0169-5347, https://doi.org/10.1016/j.tree.2005.04.004.
[5] Sah, P., Singh, L.O., Clauset, A. *et al.* Exploring community structure in biological networks with random graphs. *BMC Bioinformatics* 15, 220 (2014). https://doi.org/10.1186/1471-2105-15-220
[6] Trudeau, Richard J. (1993). *Introduction to Graph Theory* New York: Dover Pub. p. 19.
[7] Sylvester, James Joseph (1878) "Chemistry and Algebra". *Nature*. 17 (432): 284. doi:10.1038/017284a0
[8] Chartrand, Gary (1977). Introductory Graph Theory. Courier Corporation.
[9] Satyanarayana, Bhavanari; Prasad, Kuncham Syam, Discrete Mathematics and Graph Theory, PHI Learning Pvt. Ltd., p. 460; Brualdi, Richard A. (2006), Combinatorial Matrix Classes, Encyclopedia of Mathematics and Its Applications, Cambridge University Press, p. 51.
[10] Bang-Jensen & Gutin (2000) p. 19 in the 2007 edition; p. 20 in the 2nd edition (2009).
[11] Fred E. Szabo (2015), The Linear Algebra survival Guide (1st Edition) p. 11-16.
[12] Harary, Frank (1955), "On the notion of balance of a signed graph", Michigan Mathematical Journal, 2: 143–146, Mathematical Reviews: 0067468, archived from the original on 2013-04-15
[13] Cartwright, D.; Harary, Frank (1956). "Structural balance: a generalization of Heider's theory" (PDF). Psychological Review. 63 (5): 277–293. doi:10.1037/h0046049.
[14] Patrick Sole and Thomas Zaslavsky (1994) A Coding Approach to Signed Graphs , SIAM J. DISC. MATH Vol 7, No. 4, pp. 544-553, Society for Industrial and Applied Mathematics
[15] Steven Strogatz (2010), The enemy of my enemy
[16] Becker, Hila, A Survey of Correlation Clustering
[17] Brendan J. Frey; Delbert Dueck (2007). "Clustering by passing messages between data points" Science. doi:10.1126/science.1136800.
[18] Dingyin Xia, Fei Wu, Xuqing Zhang, Yueting Zhuang : Local and global approaches of affinity propagation clustering for large scale data , J Zheijang Univ Sci A 2008 , doi: 10.1631/jzus.A0720058 , arXiv:0910.1650
[19]Jian Yu, Caiyan Jia: Convergence Analysis of Affinity Propagation, International Conference on Knowledge Science, Engineering and Management, KSEM 2009: Knowledge Science, Engineering and Management pp 54-65 , Part of the Lecture Notes in Computer Science book series (LNCS, volume 5914)
[20] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AffinityPropagation.html

[21] Kriegel, Hans-Peter; Schubert, Erich; Zimek, Arthur (2016). "The (black) art of runtime evaluation: Are we comparing algorithms or implementations?". *Knowledge and Information Systems*. p. 341–378. doi:10.1007/s10115-016-1004-2. Semantic Scholar 40772241.
[22] MacQueen, J. B. (1967). *Some Methods for classification and Analysis of Multivariate Observations*. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press. p. 281–297. Mathematical Reviews: 0214227. Zentralblatt MATH: 0214.46201.
[23] Steinhaus, Hugo (1957). "Sur la division des corps matériels en parties". *Bull. Acad. Polon. Sci.* (in French). 4 (12): 801–804. Mathematical Reviews 0090073. Zentralblatt MATH: 0079.16403.
[24] Lloyd, Stuart P. (1957). "Least square quantization in PCM". *Bell Telephone Laboratories Paper*. Lloyd, Stuart P. (1982). "Least squares quantization in PCM"(PDF). *IEEE Transactions on Information Theory*. p. 129–137. CiteSeerX 10.1.1.131.1338. doi:10.1109/TIT.1982.1056489
[25] Forgy, Edward W. (1965). "Cluster analysis of multivariate data: efficiency versus interpretability of classifications". *Biometrics*. P. 768–769. JSTOR 2528559.
[26] Kriegel, Hans-Peter; Schubert, Erich; Zimek, Arthur (2016). "The (black) art of runtime evaluation: Are we comparing algorithms or implementations?". *Knowledge and Information Systems*. p. 341–378. doi:10.1007/s10115-016-1004-2, Semantic Scholar 40772241.
[27] David Arthur and Sergei Vassivitsky , k-means ++: the advantages of careful seeding https://theory.stanford.edu/~sergei/papers/kMeansPP-soda.pdf
[28] Lloyd, Stuart P. (1982), "Least squares quantization in PCM", *IEEE Transactions on Information Theory*, p. 129–137, doi:10.1109/TIT.1982.1056489.
[29] Du, Qiang; Faber, Vance; Gunzburger, Max (1999), "Centroidal Voronoi tessellations: applications and algorithms", *SIAM Review*, p. 637–676, doi:10.1137/S0036144599352836
[30] Celebi, M. E.; Kingravi, H. A.; Vela, P. A. (2013). "A comparative study of efficient initialization methods for the *k*-means clustering algorithm". *Expert Systems with Applications*. p. 200–210. arXiv:1209.1960. doi:10.1016/j.eswa.2012.07.021.
[31] Arthur, David; Manthey, B.; Roeglin, H. (2009). "k-means has polynomial smoothed complexity" *Proceedings of the 50th Symposium on Foundations of Computer Science (FOCS)*. arXiv:0904.1113
[32] Arthur, David; Vassilvitskii, Sergei (2006-01-01). *How Slow is the* k*-means Method?*. *Proceedings of the Twenty-second Annual Symposium on Computational Geometry*. SCG '06. New York, NY, USA: ACM. pp. 144–153. doi:10.1145/1137856.1137880.
[33] Kulis, Brian; Jordan, Michael I. (2012-06-26). *Revisiting* k*-means: new algorithms via Bayesian nonparametrics* (PDF). *ICML*. pp. 1131–1138. ISBN 9781450312851
[34]Biggs, Norman (1993), Algebraic Graph Theory, Cambridge Mathematical Library (2nd ed.), Cambridge University Press, Definition 2.1, p. 7
[35]Harary, Frank (1962), "The determinant of the adjacency matrix of a graph", SIAM Review, 4 (3): 202–210, doi:10.1137/1004057, MR 0144330.
[36] Shum, Kenneth; Blake, Ian (2003-12-18). "Expander graphs and codes". Volume 68 of DIMACS series in discrete mathematics and theoretical computer science. Algebraic Coding Theory and Information Theory: DIMACS Workshop, Algebraic Coding Theory and Information Theory. American Mathematical Society. p. 63.

[37] Borgatti, Steve; Everett, Martin; Johnson, Jeffrey (2018), Analyzing Social Networks (2nd ed.), SAGE, p. 20
[38] Newman, Mark (2018), Networks (2nd ed.), Oxford University Press, p. 110
[39] Loet Leydersdorff and Liwen Vaughan , Co-occurrence Matrices and their Applications in Information Science: Extending ACA to the Web Environment , Journal of the American Society for Information Science and Technology (JASIST) , https://www.leydesdorff.net/aca/#_ftnref1
[40]M. Kessler. Bibliographic coupling between scientific papers.
[41]H. Small. Co-citation in the scientific literature: A new measure of the relationship between documents. Journal of the American Society for Information Science, doi: https://doi.org/10.1002/asi.4630240406
[42] Venu Satuluri and Srinivasan Parthasarathy , Symmetrizations for Clustering Directed Graphs EDBT/ICDT '11: Proceedings of the 14th International Conference on Extending Database Technology, doi: https://doi.org/10.1145/1951365.1951407
[43] Venu Satuluri , Srinivasan Parthasarathy (2011), "Symmetrizations for clustering directed graphs", EDBT 2011, 14th International Conference on Extending Database Technology, Uppsala, Sweden, March 21-24, 2011, DOI: 10.1145/1951365.1951407
[44] Ng, A.Y.; Jordan, M.I.; Weiss, Y. (2001), "On Spectral Clustering: Analysis and an Algorithm" (PDF), *Advances in Neural Information Processing Systems*, MIT Press, p. 849–856
[45] Li, Xin-Ye; Guo, Li-Jie (2012), "Constructing affinity matrix in spectral clustering based on neighbor propagation", *Neurocomputing*, p.: 125–130, doi:10.1016/j.neucom.2012.06.023
[46] Networkx Documentation, https://networkx.org/documentation/stable
[47] Matplotlib Documentation, https://matplotlib.org/3.3.3/contents.html
[48] Numpy Documentation, https://numpy.org/doc
[49] Scikit-learn Documentation, https://scikit-learn.org/stable
[50] Erdős, P.; Rényi, A. (1959). "On Random Graphs." *Publicationes Mathematicae*. p. 290–297
[51] Gilbert, E.N. (1959). "Random Graphs" *Annals of Mathematical Statistics*. p. 1141–1144. doi:10.1214/aoms/1177706098
[52] Newman, Mark. E. J.; Strogatz, S. H.; Watts, D. J. (2001). "Random graphs with arbitrary degree distributions and their applications". *Physical Review E*. 64: 026118. arXiv:cond-mat/0007235 doi:10.1103/PhysRevE.64.026118