UNIVERSITY OF TURIN
Doctoral School of Sciences and Innovative Technologies
Ph.D. Program in Computer Science

Substructures in multilayer, temporal, and signed networks: methods and applications

Advisor:
Prof. Giancarlo Ruffo
Co-advisor:
Dr. Francesco Bonchi

Ph.D. Dissertation of:
Edoardo Galimberti

Cycle XXXII

Abstract

Networks are a powerful paradigm to model complex systems. Specific network models enrich the standard network representation with additional information to better capture real-world phenomena. For instance, multilayer networks allow multiple edges of various types among the same set of entities. Temporal networks, that represent how the relations between the entities of a system are established/broken along time, and signed networks, that report whether each edge interaction is positive or negative, can be considered special cases of multilayer networks. Despite the keen interest in a variety of problems, algorithms, and analyses, these types of networks together with the unprecedented increasing availability of data still hold new challenges and opportunities.

Extracting dense structures from complex networks has emerged as a key graph-mining primitive in a variety of scenarios. In the first part of this thesis, we present advancements to the state of the art by the introduction of novel definitions and algorithms for the extraction of dense structures from multilayer, temporal, and signed networks. Moreover, in the second part, we show how the study of the substructures of temporal and signed networks can lead to the finding of valuable insights about real-world phenomena.

This thesis has been revised and positively evaluated, considering it admissible to the defense, by **Matteo Riondato** (Amherst College, Amherst, USA), and **Matteo Magnani** (Uppsala University, Uppsala, Sweden).

Contents

Abstract **i**

1 Introduction **1**

2 Background and related work **5**
- 2.1 Network theory . . . 5
- 2.2 Related work . . . 8

I Methods 15

3 Core decomposition in multilayer networks **17**
- 3.1 Multilayer core decomposition and related problems . . . 19
- 3.2 Algorithms for multilayer core decomposition . . . 22
- 3.3 Algorithms for inner-most multilayer cores . . . 33
- 3.4 Multilayer densest subgraph . . . 37
- 3.5 Multilayer quasi-cliques . . . 42
- 3.6 Multilayer community search . . . 45
- 3.7 Summary . . . 47

4 Core decomposition in temporal networks **49**
- 4.1 Temporal core decomposition . . . 51
- 4.2 Computing all span-cores . . . 53
- 4.3 Computing maximal span-cores . . . 55
- 4.4 Temporal community search . . . 57
- 4.5 Experiments . . . 64
- 4.6 Summary . . . 71

5 Polarized communities in signed networks **73**
- 5.1 2-Polarized-Communities . . . 75
- 5.2 Algorithms for 2-Polarized-Communities . . . 76
- 5.3 Hardness and tightness . . . 79
- 5.4 Experiments . . . 81
- 5.5 Summary . . . 87

II Applications 89

6 Measures and patterns of the scientific migration network **91**
- 6.1 Scientific migration network . . . 92
- 6.2 Hubs and authorities . . . 94
- 6.3 Local patterns . . . 104
- 6.4 Case studies . . . 106
- 6.5 Summary . . . 110

7 Span-cores in face-to-face interaction networks 111
7.1 Temporal patterns . . . 111
7.2 Anomaly detection . . . 114
7.3 Graph embedding and (supervised) vertex classification . . . 115
7.4 Summary . . . 117

8 Visualizing structural balance in signed networks 119
8.1 Structural-balance-viz . . . 120
8.2 Validation and application . . . 123
8.3 Summary . . . 126

9 Conclusions 127

Chapter 1

Introduction

“*Complex system*” is a terminology widely used to define a set of entities that interact with each other in a variety of different ways. So far, researchers could not converge to a unique definition of complex system given the vast diversity of systems that are defined as such [8]. Nonlinear interactions are one of the greatest challenges in the study of complex systems and the reason of the emergence of states that are not mere combinations of the states of the individual units comprising the system. Complex systems are extremely pervasive; examples are the brain, the immune system, biological cells, ant colonies, the Internet and the World Wide Web, the stock market, and the society. The modern study of complex systems aims to understand how interactions give rise to behavioral patterns, the process of formation of complex systems through pattern formation and evolution, and ideal ways of describing complex systems [198].

Complex networks are the main instrument for describing and studying complex systems. The simplest network (or *graph*) model is composed of two elements: a set of nodes and a set of edges which represent the entities and the interactions occurring between the entities, respectively[1]. By such a simple representation, researchers are able to model a lot of variegated phenomena. For instance, our brain is a network in which nodes are regions of interest and links map functional correlations observed during fMRI (functional magnetic resonance imaging) scans. The World Wide Web is probably the most popular example of network: hyperlinks (edges) connect the web pages (vertices). Larry Page et al. [187] based the first algorithm for ranking web pages in the Google search engine on the structure of the web graph. Again, many graph-analysis methods have been proposed to study spreading processes of diseases, e.g., seasonal influenza, in contact networks where humans are represented by vertices while edges report proximity interactions [105].

The need to obtain better understanding of complex systems summed with their inherent complexity are factors that explain the increasing interest in enhancing complex-networks analytics tools and algorithms. The growing diffusion of the Internet, the increasing pervasiveness of personal and wearable devices, and the rising of the Internet of Things provide the possibility to access unprecedented amounts of data and to exploit them to unveil additional findings about complex systems. Moreover, each piece of information may be extremely rich and detailed: when we record an interaction we often know by which means it happened (e.g., by phone call or text messages), the time when it occurred, and whether it had positive or negative acceptation (e.g., a smiling emoji or a sad one).

These changes led to the introduction of network models that enrich the standard network representation with additional features to capture novel and interesting structural properties of the phenomena under analysis. *Multilayer networks* [142] model complex systems where various types of relations might occur among the same set of entities; *temporal networks* [127] are representations of entities, their relations, and how these relations are established/broken along time; and, *signed networks* represent networked data where edge annotations express whether each edge interaction is friendly (positive) or antagonistic (negative). Both temporal and signed networks can be seen as special cases of multilayer networks. In the first case, the snapshots of a temporal network correspond to the layers of a multilayer network ordered in time. In the latter case, signed networks

[1]We use the terms “network” and “graph” interchangeably throughout the thesis. Analogously, we will interchange the usage of “node” and “vertex” , and of “edge” and “link”, unless otherwise specified.

are composed of two layers, one positive and one negative; in addition, positive and negative interactions are not allowed between the same pair of entities.

Such an exciting mix of exceptional data sources and of complex models presents new challenges and opportunities. On the one hand, some of key graph-mining primitives, e.g., the identification of dense structures in networks [156], are not directly applicable to these more complicated network models and would prevent us to fully exploit of their characteristics. How can the definition of a dense structure take into account multiple layers of interactions? Which are the substructures of interest in a graph with positive and negative relations? Therefore, we are called to design novel definitions of the ordinary graph-mining primitives and the corresponding algorithms to take full advantage of these types of complex networks. On the other hand, the abundance of information allows us to carry out more sophisticated and in-depth analysis of real-world events. For example, by temporal networks, we are able to study evolving phenomena, e.g., the evolution of a migration process or the behavioral patterns of children during a school day.

In this thesis, we at first propose a set of novel definitions of dense structures in multilayer, temporal, and signed networks with particular attention to the algorithmic aspects. Then, we investigate specific applications of such network paradigms to concrete studies of real-world datasets.

Contributions and structure of the thesis

Extracting dense structures from large graphs has emerged as a key graph-mining primitive in a variety of scenarios [156], ranging from web mining [110], to biology [91, 154], and finance [72]. Dense structures can help in studying contact networks among individuals to quantify the transmission opportunities of respiratory infections [107]. Anomalously dense structures among entities in a co-occurrence network have been used to identify, in real-time, events and buzzing stories [12, 44]. Peculiar dense structures are often associated to polarization and unbalance in online debate networks [59]. Although the literature about complex networks has recently grown extremely fast, the problem of identifying dense structures in specific types of networks has, surprisingly, still unexplored facets.

Among the many definitions of dense structures, *core decomposition* plays a central role. The *k-core* of a graph is defined as a maximal subgraph in which every vertex is connected to at least k other vertices within such subgraph. The set of all k-cores of a graph G forms the *core decomposition* of G [208]. The importance of core decomposition relies in the fact that it can be computed in linear time [30], and can be used to speed-up/approximate dense-subgraph extraction according to various other definitions [78, 148, 10, 123]. In addition, it has been recognized as an important tool to analyze and visualize complex networks in several domains [7, 29].

Orthogonal approaches focus on the identification of a single dense portion of a complex network. Extracting dense subgraphs according to the average degree (i.e., two times the number of edges divided by the number of vertices) has attracted most of the research in the field since it is solvable in polynomial time [114]. This problem is commonly referred to as the *densest subgraph*. Many of other notions of density have been proposed in literature, most of which have been shown to be **NP**-hard [227, 234].

A different bulk of literature deals with methods to find many communities, i.e., dense structures, while partitioning the whole complex network [89]. Among the many approaches, of particular interest is the one provided by the *correlation clustering* framework [24] for signed networks, which asks to partition the vertices into communities so as to maximize (minimize) the number of edges that "agree" ("disagree") with the partitioning, i.e., the number of positive (negative) edges within clusters plus the number of negative (positive) edges across clusters.

The first purpose of the present thesis is to introduce novel definitions and algorithms for the extraction of dense structures from multilayer, temporal, and signed networks. Specifically, in Part I, we generalize the concept of core decomposition to multilayer networks and we show a series of applications built on top of it; then, we define temporal core decomposition in temporal networks; and, finally, in the context of discovering polarization in large-scale online data, we study the problem of identifying polarized communities in signed networks. The specific contribution of each chapter of Part I is summarized as follows:

- **Chapter 3** studies the problem of *core decomposition in multilayer networks* and presents a series of applications built on top of it. We first focus on the problem of finding only the *max-*

imal cores, i.e., the densest ones, in the decomposition. Then, we show how multilayer core decomposition finds application to the problem of *densest-subgraph extraction from multilayer networks* and in the speed-up the extraction of *frequent cross-graph quasi-cliques* [137]. Finally, we show how multilayer core decomposition can be used to generalize the *community-search* problem [215] to the multilayer setting. The original contributions of this chapter have been published in the *Proceedings of the 26th ACM International Conference on Information and Knowledge Management (CIKM 2017)* [96] and have been accepted for publication in the *Journal of ACM Transactions on Knowledge Discovery from Data (TKDD)* [97].

- **Chapter 4** introduces the definition of *span-core decomposition in temporal networks* where span-cores are dense structures, based on minimum degree, during an interval of contiguous timestamps. Also in this case, we investigate the problem of identifying the *maximal span-cores* only. Finally, we introduce the problem of *temporal community search* [215]. The original contributions of this chapter have been published in the *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM 2018)* [95] and are currently submitted for review [98].

- **Chapter 5** proposes the 2-Polarized-Communities problem which requires finding a dense structure composed two warring (i.e., polarized) communities in a signed network. Our hypothesis is that such 2-community polarized structure accurately captures controversial discussions in real-world social-media platforms. The original contributions of this chapter have been published in the *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM 2019)* [45].

The design of ad-hoc definitions and algorithms for the identification of specific substructures in complex networks is only the first step towards gaining better understanding of complex systems. Of fundamental importance is finding reliable data sources that can describe phenomena of interest, identifying the network representation that better models the characteristics of the data, and apply the most suitable methods for the extraction of valuable insights.

For instance, the migration of humans can be considered a complex system where different locations on the Earth interact by exchanging individuals with the others. It has shown to be crucial importance in modern history [185, 146]. Among the many types of migrations, the mobility of researchers, scientists, and academics is fundamental since it moves knowledge, ideas, and information, which are considered to be among the major economic production factors in today's economy [177]. Fortunately, data obtained from online academic platforms, such as ORCID [43], can be modeled as complex networks to derive specific insights about how academics move between countries, e.g., which role each state plays in the system.

On a smaller scale, people gather in public environments, such as schools, workplaces, and conferences. The network representation has proved useful to understand the structure of the systems of interactions between individuals and to describe processes occurring in these systems [26]. Each face-to-face relation is now easy to record thanks to non-obtrusive sensing systems and infrastructures. In particular, the study of mixing patterns of children in school environments has resulted to be of extreme interest because they can help to quantify the transmission opportunities of respiratory infections and to identify situations within schools where the risk of transmission is higher [176, 113].

Finally, polarization around controversial issues is rapidly growing due to the scale and speed of discussions enabled by modern large-scale social-media platforms. Even in this case, complex (signed) networks are able to depict such phenomena [151] to ease its understanding by the application of *balance theory* [124]. *Network visualization* can be powerful in this scenario, and in many others, to complement standard network analysis techniques since it is extremely effective in communicate findings [150].

The second aim of this thesis is to show how temporal and signed networks can be exploited to carry out analysis of real-world data to derive interesting insight about the substructures in the corresponding complex systems. In the details, in Part II, we study international migrations of researchers employing data coming from ORCID; we then apply the tools introduced in Chapter 4 for the analysis of face-to-face interaction networks recorded in schools; in the final chapter, we propose a network visualization technique to highlight balanced/polarized structures in signed networks. The contributions of each chapter of Part II are the following:

- **Chapter 6** studies the international migration of researchers with the aim of identifying measures and substructures describing the countries that play a central role in such phenomenon. We define the *scientific migration network* from ORCID data, where nodes represent world countries and edges account for a migratory flow from a country to another, and employ the HITS algorithm [143] to find which countries are *hubs* (providers) and *authorities* (attractors). Moreover, we investigate local patterns and characteristics of the neighborhood of hubs and authorities to derive the motivations behind the HITS algorithm. The original contributions of this chapter are currently submitted for review [229].
- **Chapter 7** complements Chapter 4 in the study of temporal networks. Specifically, we exploit the concept of (maximal) span-core for deriving temporal patterns, detecting anomalous interactions, and classifying individuals in face-to-face interaction networks gathered in schools. As for Chapter 4, the original contributions of this chapter have been published in the *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM 2018)* [95] and are currently submitted for review [98].
- **Chapter 8** proposes a novel method for visualizing signed networks whose purpose is to identify balances/polarized structures in real-world data modeling discussions, e.g., in online social media or political debates. The core of the visualization relies on balance theory, similarly to Chapter 5. The original contributions of this chapter have been accepted for publication in the *Proceedings of the 8th International Conference on Complex Networks and Their Applications (COMPLEX NETWORKS 2019)* [99].

Chapter 2 introduces the background about network analysis and the work related to the content of this thesis. Chapter 9 concludes the thesis and proposes interesting avenues for future work.

Chapter 2

Background and related work

In this chapter we first introduce some background about network theory (Section 2.1), with particular emphasis on measures of density and centrality, which are the topics mostly related to the proposed works. Then, we describe the bunches of works related to the contents of this thesis (Section 2.2).

2.1 Network theory

We define a standard network by a pair $G = (V, E)$ consisting of a set of vertices V and a set of edges $E \subseteq V \times V$. We say that two vertices $u, v \in V$ are *connected*, equivalently *adjacent* or *neighbors*, if edge (u, v) belongs to E.

In general, graphs may have peculiar features and may carry specific information over the edges (as well over the vertices). For example, graphs can be partitioned into two main categories on the basis of whether the edges are *undirected* or *directed*. Given two vertices $u, v \in V$ of an undirected graph, the edge (v, u) is equal to the edge (u, v), and it has not a direction associated whit it. Differently, for a directed graph the edge (v, u) is distinct from the edge (u, v), and it has a direction associated with it. In this thesis we consider both undirected and directed networks.

As introduced beforehand, graphs' edges might be enriched by different types of information. Often we want to assign a *weight* to each edge; then, we refer to a weighted graph by a triplet $G = (V, E, w)$ where $w : E \to \mathbb{R}$ is a weighting function that assigns a weight to each edge. Weights may be restricted to be rational or integer numbers, or to be positive. A special case of a weighted graph is a *signed graph* or *signed network*, which is a network whose edges are either positive or negative, or, equivalently, have weight $+1$ or -1. Signed graphs are usually referred to $G = (V, E_+, E_-)$ where E_+ is the set of positive edges and E_- the set of negative edges.

Additional complex network paradigms allow multiple edges between the same pair or vertices. For example, *multilayer networks* [142] model complex systems where multiple interactions of different types may exist between any pair of entities. A multilayer graph is defined as $G = (V, E, L)$ where L is a set of layers and $E \subseteq V \times V \times L$ is a set of labeled edges; then, in this case edges are represented by triplets composed of two vertices and a layer. Similar to multilayer networks, *temporal networks* [127] represent systems whose topology changes over time. Instead of having a layer, temporal edges are labeled by a *timestamp* which assigns a temporal collocation. We denote a temporal network as $G = (V, T, \tau)$ where $T = [0, 1, \ldots, t_{max}] \subseteq \mathbb{N}$ is a discrete time domain, and $\tau : V \times V \times T \to \{0, 1\}$ is a function defining for each pair of vertices $u, v \in V$ and each timestamp $t \in T$ whether edge (u, v) exists in t. The main difference between multilayer graphs and temporal networks is that layers are not ordered while timestamps follow the temporal ordering.

Examples. A standard undirected graph might depict, e.g., a static network of friends, where each edge represents a relationship of friendship between two persons, which is bidirectional. Directed networks are used in other context where the direction of the connections matters, e.g., in a citation network of academics, in which an edge $(u, v) \in E$ indicates that the researcher u cited a work of the researcher v but no viceversa. Weighted networks are often employed to represent supply chains since edge weights can include information about the amount (or the worth) of the goods

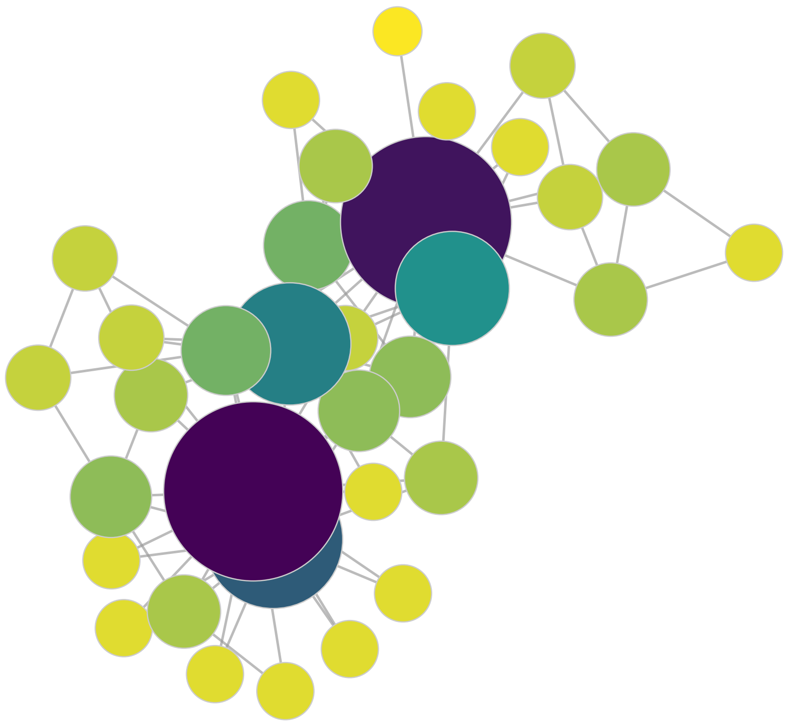


Figure 2.1: Zachary's karate club network: a social network of friendships between 34 members of a karate club in a US University in the 70s. Nodes' size and color (on a scale from yellow to blue) is according to the number of their connections.

moved from a point to another of the supply. Social networks in which relations can be either friendly or antagonistic, e.g., a debate on Twitter, are represented by signed networks since they can differentiate positive and negative interactions. Biological systems are extremely complex. For example, proteins communicate between each other in many different ways and by different means. Therefore, biologists prefer multilayer networks to study protein-to-protein interactions. Temporal networks find application in a variety of contexts where the topology of the system changes over time. They have been recently applied to the study of face-to-face contact networks for the understanding of respiratory diseases.

In general, the types of systems that we can model and study by means of complex networks is endless; even in very small environments, as in the real-world case shown in Figure 2.1 representing a social network, we are able to derive complex patterns in the structure of the interpersonal relations, i.e., the presence of two individuals more central (important) than the others and the division of the vertices into two clusters [244].

2.1.1 Dense structures

In almost every kind of network, *density* is an indication of importance of certain regions of the graph. Dense structures in a network may indicate high degrees of interaction, or mutual similarity and hence collective characteristics, attractive forces, favorable environments, or critical mass. From a theoretical point of view, dense structures have many interesting properties. They naturally have small diameter[1], therefore routing within these components is rapid. Dense regions are also robust, in the sense that many connections can be broken without splitting the component [156]. A less known but equally important property of dense structure comes from percolation theory: if a graph is sufficiently dense, then there is very high probability of propagating a message across the whole graph [117].

Various definitions of dense structures have been explored in different studies. Generally, they

[1]We define the *diameter* of a graph as the greatest distance, in term of edges, between any pair of vertices.

are based on two different concepts of density, i.e., *absolute density* and *relative density.* An absolute density measure establishes the conditions for what constitutes a dense pattern independently of what is outside the patterns itself. The definition of k-core is an example: a subgraph is a k-core if and only if every vertex is connected to at least k other vertices. On the other hand, a relative density measure has no preset parameters for deciding whether a structure is sufficiently dense or not; rather, it compares the density of one region of the graph to another, with the goal of finding the densest regions. In order to set the boundaries of such structures, a metric is used with the aim of maximizing the difference between intra-component density and inter-component density [156]. Cut (i.e., the number of edges between a component and what is outside the component) minimization is an example of objective used for this purpose.

Degree-based densities. In literature, the most-in-use definitions of density are *average degree* and *minimum degree.* Given a standard undirected graph $G = (V, E)$ and a subset of vertices $S \subseteq V$, we denote the degree of a node $u \in S$ in the induced subgraph $G[S] = (S, E[S])$ as $deg_S(u) = \{v \in S \mid (u, v) \in E[S]\}$, i.e., the number of neighbor nodes of u in $G[S]$. As a consequence, the average degree of a subgraph $G[S]$ is

$$\operatorname*{avg}_{u \in S} deg_S(u). \tag{2.1}$$

We analogously define the minimum degree of $G[S]$ as

$$\min_{u \in S} deg_S(u). \tag{2.2}$$

In the following section, we will dive deeper into the algorithms that find dense structures based on average degree and minimum degree, and variations of such densities.

Cliques and quasi-cliques. The densest structure that we can find in a graph is a *clique.* A subset of vertices $S \subseteq V$ in a graph G represents a clique if each pair of vertices in S is connected by an edge in the induced subgraph $G[S] = (S, E[S])$; or, equivalently, $deg_S(u) = |S| - 1$ for each node $u \in S$, where $deg_S(u)$ indicates the degree of u in $G[S]$. The definition of clique is quite strict and cliques of remarkable dimension are rare to find in real-world complex networks. For this reason, many relaxations of the clique definition have been proposed in terms of density, degree, or distance. The most common one is the concept of *quasi-clique* in which a parameter $\gamma \in [0, 1]$ defines how close the quasi-clique is to be a clique. A subset of vertices $S \subseteq V$ is a γ-quasi-clique if $G[S] = (S, E[S])$ has at least $\gamma\binom{|S|}{2}$ edges [1] or, alternatively, each vertex in S has at least degree $\gamma(|S| - 1)$ [175].

Minimum-degree-based dense structures. The most used definitions of dense structure based on minimum degree are *core* and *plex.* A k-core (or a core of order k) is a maximal subset of vertices $S \subseteq V$ of G such that each node $u \in S$ has $deg_S(u) > k$ [208]. Similarly, S is a *k-plex* if each vertex in S has at least degree $|S| - k$ in $G[S]$ [209]. The parameter $k \in \mathbb{N}$ indicates the importance of cores and plexes: the higher k the more dense and cohesive a core is; viceversa for plexes.

Distance-based dense structures. Other definitions of dense structure are build upon the concept of distance and, in particular, of shortest path[2]. Given a graph $G = (V, E)$ and an integer number k, the subset of vertices $S \subseteq V$ is a *kd-clique* if the shortest path from any vertex to any other of S is no more than k. In this case, paths may go outside $G[S]$, i.e., the subgraph induced by S, and include any vertices and edges of G [168]. The definition of *k-club* is exactly the same; however, in this case, paths may not go outside $G[S]$ [178]. Again, $k \in \mathbb{N}$ is an indication of how cohesive a core or a club is: the shorter shortest paths are, the more the structures are cohesive and relevant.

Related works about cores and core decomposition, and other types of dense structure in different kinds of networks are discussed in greater detail in Section 2.2.

2.1.2 Centrality measures

In many applications, researchers and practitioners are not only interested in studying the important regions of a network, rather they want to understand the role that specific entities (or

[2]The shortest path between two vertices $u, v \in V$ in a graph $G = (V, E)$ is the path such that the number of its constituent edges is minimized.

connections) play with respect to the global topology of the graph. The importance of a node (or an edge) is assessed by *centrality measures* [92], that may be defined on several structural features of the node, like its connectivity or its position with respect to the other vertices.

Degree-based centrality measures. The simplest and most commonly used centrality measure is *degree centrality.* For a vertex $u \in V$ in a standard graph $G = (V, E)$, it is defined as its degree:

$$c_d(u) = deg(u). \tag{2.3}$$

Distance-based centrality measures. Other centrality measures rely upon the concepts of distance/proximity and, as a consequence, of shortest path. *Closeness centrality* [31] has been defined from the intuition that a node is more central when it is closer to all other nodes. The closeness centrality of a node $u \in V$ is defined by the following ratio:

$$c_c(u) = \frac{1}{\sum_{v \in V} d(u, v)}, \tag{2.4}$$

where $d(u, v)$ is the length of the shortest path between nodes u and v. Similar is the definition of *betweenness centrality* [92] which counts the number of times a node acts as a bridge along the shortest path between two other nodes. The betweenness centrality of a node u is

$$c_b(u) = \sum_{\substack{v,z \in V \\ u \neq v \neq z}} \frac{\sigma_{vz}(u)}{\sigma_{vz}}, \tag{2.5}$$

where σ_{vz} is the total number of shortest paths from node v to node u, and $\sigma_{vz}(u)$ is the number of such paths passing through node u. Finally, *Katz centrality* [139] is a generalization of degree centrality. While degree centrality measures the number of direct neighbors of a vertex u, Katz centrality takes into account the number of nodes in the graph that can be connected through a path from u. The contribution of nodes distant from the origin u are penalized by an attenuation factor.

PageRank. *PageRank* centrality [187] is at the basis of the algorithms used by Google for ranking web pages in their search engine. Differently than the centrality measures described above, which are usually employed in undirected graphs, it is defined for directed networks. Since its introduction, it has found application in many areas and many variations have been proposed, e.g., [190]. The assumption at the basis of PageRank is that the quality of the edges adjacent to a node is more important than their quantity for defining the importance of the node itself; and, the quality of an edge is defined as the PageRank centrality of the node at the other endpoint of the edge.

HITS. Another edge-based method for ranking vertices of a graph is the HITS algorithm, introduced by Kleinberg [143]. HITS computes two scores (centralities) for each node, i.e., the *hub score* and the *authority score.* The underlying idea is similar to PageRank: nodes of high hub score have many outgoing edges towards vertices of high authority score, while authorities have many incoming edges from hubs. The details of (slightly modified versions of) PageRank and HITS are reported in Chapter 6 (Section 6.2) in which we employ different centrality measures to propose a network-driven study of mobility/migration of scientists across the world countries.

2.2 Related work

2.2.1 Core decomposition

Here we recall the concept of core as a dense structure and we introduce the definition of *core number* of a vertex and *core decomposition* in standard graphs.

Definition 2.1 (k-core, core number, and core decomposition)**.** *The* k-core *(or core of order k) of a standard graph $G = (V, E)$ is a* maximal *set of vertices $C_k \subseteq V$ such that $\forall u \in C_k : deg_{C_k}(u) \geq k$. The* core number *(or* core index*) of a vertex u is the highest order of a core that contains u. The set of all k-cores $V = C_0 \supseteq C_1 \supseteq \cdots \supseteq C_{k^*}$ ($k^* = \arg\max_k C_k \neq \emptyset$) is the* core decomposition *of G.*

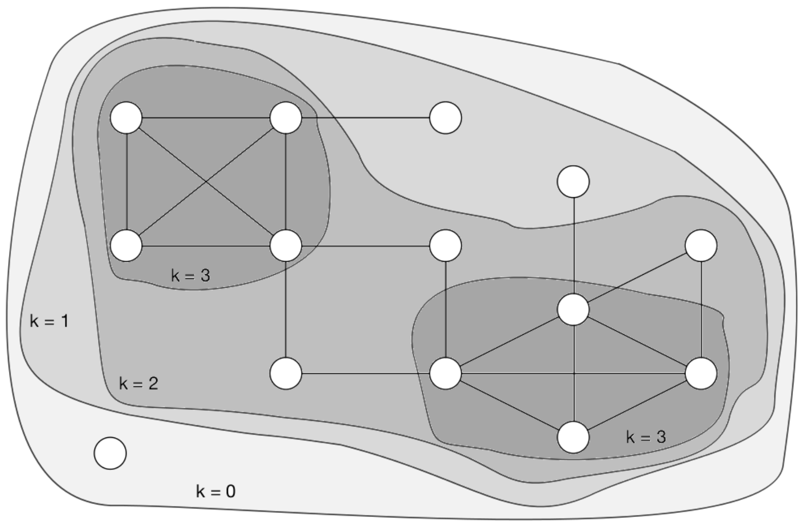


Figure 2.2: Example of core decomposition of a standard graph. The graph has 4 cores: the 0-core is the entire graph, while the 3-core includes two connected components.

Core decomposition can be computed in linear time by iteratively removing the smallest-degree vertex and setting its core number as equal to its degree at the time of removal [30]. Among the many definitions of dense structures, core decomposition is particularly appealing as, among others, it is fast to compute, and can speed-up/approximate dense-subgraph extraction according to various other definitions. For instance, core decomposition allows for finding cliques more efficiently [78], as a k-clique is contained into a $(k-1)$-core, which can be significantly smaller than the original graph. Moreover, core decomposition is at the basis of approximation algorithms for the densest-(at-least-k-)subgraph problem [148, 10], and betweenness centrality [123]. Core decomposition has also been recognized as an important tool to analyze and visualize complex networks [29, 7] in several domains, e.g., bioinformatics [20, 239], software engineering [246], and social networks [141, 102]. It has been studied under various settings, such as distributed [180], streaming/maintenance [205, 161], and disk-based [57], and generalized to various types of static graphs, such as uncertain [47], directed [109], weighted [101, 75], bipartite graphs [165], or including attributes on the nodes [245]. For a comprehensive survey about theory, algorithms, and applications of core decomposition we refer to [171].

In Chapter 3 of this thesis we adopt the definition of multilayer core by Azimi-Tafreshi et al. [19] to study how to efficiently compute the complete core decomposition of multilayer networks. In [19] a core identified by an $|L|$-dimensional integer vector $\vec{k}$ (with $|L|$ being the number of layers of the given multilayer network), where every component of $\vec{k}$ refers to the minimum-degree constraint required for that core in the corresponding layer (i.e., every $\vec{k}[i]$ component states the minimum degree required for that core in the i-th layer). Apart from introducing the definition of multilayer core, Azimi-Tafreshi et al. study the core-percolation problem from a physics standpoint, without providing any algorithm. Specifically, they characterize cores on 2-layer Erdős-Rényi and 2-layer scale-free networks, then they analyze real-world (2-layer) air-transportation networks.

Wu et al. [237] have proposed a core decomposition on temporal networks, which does not take any kind of temporal constraint into account. They define indeed the (k, h)-core as the largest subgraph in which every vertex has at least k neighbors and there are at least h temporal edges between the vertex and its neighbors, without any restriction on when these h edges occur: the sequentiality of connections is not taken into account and non-contiguous timestamps can support the same core. In fact, the (k, h)-core decomposition can be seen as a kind of weighted static core decomposition on the weighted static network resulting from the aggregation of the temporal network. In Chapter 4 we devise a formulation of cores for temporal networks having a clear temporal collocation and continuous spans. As we will see in Chapter 7, associating a temporal collocation to each core is important in concrete applications.

2.2.2 Community search

Given a standard graph and a set of query vertices, the *community search* problem aims at finding a cohesive subgraph containing the query vertices. Community search has attracted a great deal of attention in the last years [86, 132]. Sozio and Gionis [215] are the first to introduce this problem by employing the minimum degree as a cohesiveness measure. Their formulation can be solved by a simple (linear-time) greedy algorithm, which resembles the traditional 2-approximation algorithm for densest subgraph proposed in [73]. More recently, Cui et al. [64] devise a local-search approach to improve the efficiency of the method defined in [215], but only for the special case of a single query vertex. The case of multiple query vertices has instead been addressed by Barbieri et al. [25], who exploit core decomposition as a preprocessing step to improve efficiency. They also tackle the problem of *minimum community search*, i.e., a variant of community search where the size of the output subgraph has to be minimized.

Community search has also been studied under different names and/or settings. Huang et al. [130] introduce a community-search model based on the k-truss notion. Andersen and Lang [11] and Kloumann and Kleinberg [145] study seed set expansion in social graphs, in order to find communities with small conductance or that are well-resemblant of the characteristics of the query vertices, respectively. Other works define connectivity subgraphs based on electricity analogues [83], random walks [226], the minimum-description-length principle [6], the Wiener index [202] and network efficiency [201]. Recent approaches also introduce the flexibility of having query vertices belonging to different communities [39, 242]. Finally, community search has been formalized for attributed graphs [131, 84] and spatial graphs [85] as well.

In Chapters 3 and 4 we formulate the community-search problem for multilayer and temporal graphs, respectively. The problem statement for multilayer networks (Chapter 3) adopts the early definition by Sozio and Gionis [215] which measures the cohesiveness of a subgraph by means of its minimum degree. On the other hand, in Chapter 4, we provide a novel definition of the problem by asking for a set of subgraphs containing the given query vertices, along with their corresponding temporal collocation, such that a global density measure (based on minimum degree) of the identified subgraphs is maximized and the union of the temporal intervals spanned by those subgraphs covers the whole underlying temporal domain.

2.2.3 Densest subgraph

Several notions of density exist in the literature, each of which leading to a different version of the problem of extracting a single dense subgraph. While most variants are **NP**-hard, or even inapproximable, extracting dense subgraphs according to average degree is solvable in polynomial time [114]. As a result, such a density has attracted most of the research in the field, so that the subgraph maximizing the average-degree density is commonly referred to as the *densest subgraph*.

Goldberg [114] provides an exact solution based on iteratively solving ad-hoc-defined minimum-cut problem instances. Although principled and elegant, Goldberg's algorithm cannot scale to large graphs. Asahiro et al. [18] and Charikar [73] provide a more efficient (linear-time) $\frac{1}{2}$-approximation algorithm that is capable of handling large graphs. The algorithm greedily removes the smallest-degree vertex, until the graph has become empty. Among all subgraphs produced during this vertex-removal process, the densest one is returned as output. Note that this algorithm resembles the one used for core decomposition. In fact, it can be proved that the inner-most core of a graph is itself a $\frac{1}{2}$-approximation of the densest subgraph.

In the classic definition of densest subgraph there is no size restriction of the output. Variants of the problem with size constraints turn out to be **NP**-hard. Thus, approximation algorithms and other (mostly theoretic) results have been presented [17, 87, 16, 10]. A number of works depart from the classic average-degree maximization problem and focus on extracting a subgraph maximizing other notions of density. For instance, Tsourakakis et al. [228] resort to the notion of quasi-clique to define an alternative measure of density, while Tsourakakis [227] and Wang et al. [234] focus on notions of density based on k-cliques and/or triangles. The densest-subgraph problem has also been studied in different settings, such as streaming/dynamic context [21, 38, 76], and top-k fashion [22, 94, 183].

2.2.4 Substructures in multilayer networks

A number of recent contributions have emerged on the problem of extracting dense structure from a set of multiple graphs sharing the same vertex set, which is a setting equivalent to the multilayer one. Jethava and Beerenwinkel [136] define the *densest common subgraph* problem, i.e., find a subgraph maximizing the minimum average degree over all input graphs, and devise a linear-programming formulation and a greedy heuristic algorithm for it. Reinthal et al. [195] provide a Lagrangian relaxation of the Jethava and Beerenwinkel's linear-programming formulation, which can be solved more efficiently. Semertzidis et al. [210] introduce three more variants of the problem, whose goal is to maximize the average average degree, the minimum minimum degree, and the average minimum degree, respectively. They show that the average-average variant easily reduces to the traditional densest-subgraph problem, and that the minimum-minimum variant can be exactly solved by a simple adaptation of the classic algorithm for core decomposition. They also devise heuristics for the remaining two variants. A very recent work by Charikar et al. [56] further focuses on the minimum-average and average-minimum formulations, by providing several theoretical findings, including **NP**-hardness, hardness of the approximation (for both minimum-average and average-minimum), an integrality gap for the linear-programming relaxation introduced in [136, 195] (for minimum-average), a characterization in terms of parameterized complexity (for average-minimum).

Other contributions in this area deal with specific cases of 2-layer networks [238, 213] and with the *community-detection* problem [37, 181, 188, 51, 220, 225, 243]. Boden et al. [41] study *subspace clustering* for multilayer graphs, i.e., find clusters of vertices that are densely connected by edges with similar labels for all possible label sets. Yan et al. [241] introduce the problem of mining *closed relational graphs*, i.e., frequent subgraphs of a multilayer graph exhibiting large minimum cut. Jiang et al. [137] focus on extracting *frequent cross-graph quasi-cliques*, i.e., subgraphs that are quasi-cliques in at least a fraction of layers equal to a certain minimum support and have size larger than a given threshold. Interdonato et al. [135] are the first to study the problem of *local community detection in multilayer networks*, i.e., when a seed vertex is given and we want to reconstruct its community by having only a limited local view of the network. Finally, Zhu et al. [247] address the problem of finding the k most diversified d-coherent cores, i.e., the k subgraphs having minimum degree at least d that maximize the coverage of the vertices.

In Chapter 3 we introduce a formulation of the densest-subgraph problem in multilayer networks that trades off between high density and number of layers where the high density is observed. Moreover, we show that our formulation generalizes the minimum-average densest-common-subgraph problem studied in [56, 136, 195, 210] and our method to solve our formulation provides approximation guarantees for this definition as well. Furthermore, we show how to speed-up the problem of finding frequent cross-graph quasi-cliques [137].

2.2.5 Substructures in temporal networks

A number of works on extracting dense structures from a temporal network focus on the notion of densest subgraph introduced above. The densest common subgraph problem introduced by Jethava and Beerenwinkel [136] considers as input a set of graphs sharing the same vertex set, which can thus also be interpreted as a temporal network. The theoretical advancements [195, 56] and the variants of the problem [210] later introduced are valid for the temporal scenario al well.

Complementary works focus on variants of the densest-subgraph-discovery problem. Rozenshtein et al. study the problem of discovering dense temporal subgraphs whose edges occur in short time intervals considering the exact timestamp of the occurrences [200], and the problem of partitioning the timeline of a temporal network into non-overlapping intervals, such that the intervals span subgraphs with maximum total density [199]. Epasto et al. [76] deal with the problem of maintaining the densest subgraph in a dynamic setting.

Attention in the literature has also been devoted to densities other than the average degree. The notion of Δ-clique, as a set of vertices in which each pair is in contact at least every Δ timestamps, has been proposed in [231, 126]. Bentert et al. [35] introduce the Δ-k-plex, a relaxation of Δ-clique in which each vertex has an edge to all but at most $k-1$ vertices at least once every Δ consecutive timestamps. Li et al. [160] study the problem of finding the maximum (θ,Δ)-persistent k-core in a temporal network, i.e., the largest subgraph that is a connected k-core in all the subintervals of duration θ of a given temporal interval Δ.

A different, but still slightly related body of literature focuses on other definitions of temporal patterns, such as frequent evolution patterns in temporal attributed graphs [36, 134, 67], link-formation rules in temporal networks [49, 159], frequency-estimation algorithms for counting temporal motifs [149, 166], finding a small vertex set whose removal eliminates all temporal paths connecting two designated terminal vertices [248], finding a subgraph that maximizes the sum of edge weights in a network whose topology remains fixed but edge weights evolve over time [42, 170], and the discovery of dynamic relationships and events [65], or of correlated activity patterns [106].

The work enclosed in Chapter 4 differs from all the above ones as our notions do not correspond (or are straightforwardly reducible) to any of those temporal patterns.

2.2.6 Substructures in signed networks

The concept of *structural balance* first appears as psychological theory of balance in triangles of sentiments [124]. Signed networks are later introduced in the seminal work by Harary [121], who also generalizes the balance theory to signed networks [52]. Harary and Kabell develop a simple algorithm to test whether a given signed network is balanced [122] by enumerating the cycles in the network containing an even number of negative edges; in this case, the network can be partitioned into two clusters of nodes having only positive edges within and only negative edges in-between. A complete signed network is balanced if and only if all its triangles are balanced [73]. Akiyama et al. [5] study how to estimate the minimum number of sign changes required so that a signed network satisfies the balance property. Recent works link spectral properties of signed networks to the balance theory. Hou et al. [128] prove that a signed network is balanced if and only if the smallest eigenvalue of the signed Laplacian is 0. Moreover, [129] investigates the relationship between the smallest eigenvalue of the signed Laplacian and the level of balance of a signed network.

A fundamental problem studied in signed networks is correlation clustering [24], i.e., partition the nodes into clusters so as to maximize (minimize) the number of edges that "agree" ("disagree") with the partitioning. The 2-correlation-clustering problem [60], also known as the frustration-index problem [14], is also widely studied.

Signed graphs have also been studied in different contexts. Guha et al. [119] and Leskovec et al. [158] study *directed* signed graphs and develop *status theory*, which complements balance theory, to reason about the importance of the vertices in such graphs. Other lines of research include edge and vertex classification [54, 221], link prediction [157, 219], community detection [3, 9, 24, 218], recommendation [222], and more. A detailed survey on the topic is provided by Tang et al. [223].

A few recent works explore the problem of finding antagonistic communities in signed networks. Lo et al. consider directed graphs and search for strongly-connected positive subgraphs that are negative bi-cliques [167], which severely limits the size of the resulting communities. A relaxed variant for undirected networks was described in subsequent work [100]. Chu et al. propose a constrained-programming objective to find k warring factions [59], as well as an efficient algorithm to find local optima.

The problem we study in Chapter 5 could be seen at the intersection of correlation clustering and the problem of identifying antagonistic communities: we search for two clusters while we allow vertices not to be part of any cluster.

2.2.7 Scientific migration and network analysis

The mobility of scientists is a topic of broad interest that has been investigated in a series of works. It is studied in [108] adopting an economic point of view mixed with the traditional sociology of science. Saxenian [206] and Agrawal et al. [2] discuss about the concept of *brain drain* and argue that connections between migrant scientists and their home countries are persistent in time and might ease knowledge transfer backward. For these reasons, they call this phenomenon *brain circulation* or *brain bank*. Since reliable data sources about the topic are often problematic, Franzoni et al. [90] devise a survey with the intent of providing consistent data about cross-country researches. [177] explores how Scopus[3] can be exploited as data source for the study of international

[3] https://www.scopus.com

scientific mobility for countries with high adoption of the platform. The authors of [177] do not propose any network model, while they show quantitative metrics and general trends about the observed countries and researchers. A recent study by Verginer et al. [230] describes a method to extract mobility networks from a collection of four bibliographic data sources to characterize the mobility of scientists at city granularity.

Human migration was studied by means of complex networks in the recent past. [81] defines the *international migration network* as temporal weighted direct network having countries as nodes and whose edges represent stocks of migrants. Following up the seminal work by Fagiolo et al. [81], many other approaches are proposed with similar purposes, studying for example human migration from a multilayer perspective using data gathered from social networks [34]. A complementary work [82] correlates per-capita income and labor productivity with human migration and network centrality. It has been explored also how to build complex networks from worldwide migration flows to identify a socioeconomic indicator that explains the reasons behind the phenomenon [53]. Finally, Robinson et al. [197] propose a machine learning approach to predict long-term human mobility.

In Chapter 6 we propose novel attempt to model and analyze human migration, the migration of the scientific population in particular, by means of network analysis. We follow up the work by Bohannon and Doran [43] which claims that ORCID data can be used to survey scientific migration. [43] provides a collection of basic statistics about the dataset without deepening the temporal evolution of the phenomenon nor introducing a network approach.

2.2.8 Network visualization

Many network visualizations have been proposed in literature in order to graphically express specific characteristics, properties, and patterns of networks [69, 125, 140]. An example is reported in Figure 2.1 in which nodes' size and color highlight the degree centrality of the vertices, and nodes' position depicts a strong community structure of the network.

Force-based visualizations map an energy function to the desired layout and minimize it [140]; among the most famous, we recall the Fruchterman-Reingold [93] and the Kamada-Kawai [138] algorithms. Bastert and Matuszewski [27] propose a layout for directed networks to empathize the the flow direction. Hive plots [150] place nodes on radially oriented linear axes according to a coordinate system defined by nodes characteristics and/or network properties. Eigenvectors are exploited for visualizing networks in different works [191, 147, 48]. In particular, [152] studies the application of clustering, prediction, and visualization methods to signed networks by using the signed Laplacian and its eigenvalue decomposition. In particular, the proposed visualization wants show a clustering of the nodes.

In Chapter 8 we propose a novel visualization algorithm targeted to signed networks. We borrow the theoretical principles of Chapter 5 and [152] to highlight structural balance properties of a given signed network.

Part I

Methods

Chapter 3

Core decomposition in multilayer networks

In social media and social networks, as well as in several other real-world contexts – such as biological and financial networks, transportation systems and critical infrastructures – there might be multiple types of relation among entities. Data in these domains is typically modeled as a a graph composed of a superimposition of different layers, i.e., a graph where multiple edges of different types may exist between any pair of vertices [70, 51, 155]. In the literature different terminologies have been used to refer to graphs of this kind: *multilayer networks*, *multiplex networks*, *multi-dimensional networks*, *multirelational networks*, *multislice networks*, and more. There is no real uniformity in this regard, and the various terms may also refer to slightly different concepts. In this work we deal with networks composed of multiple layers, *with no inter-layer links*, and hereinafter use the term "*multilayer networks*" to refer to them.[1]

In this chapter we study the problem of *core decomposition in multilayer networks*: although the number of multilayer cores can be exponential in the number of layers, we devise efficient algorithms to compute the complete core decomposition. However, efficiency of the core decomposition is not enough. Given the potentially high number of cores, we need to provide the data analyst with additional tools to browse through the output, being able to focus only on the patterns of interest. The situation resembles that of the classic *association rules* and *frequent itemsets* mining: a potentially exponential output, efficient algorithms to extract all the patterns, the need to define concise summaries of the extracted knowledge, and the opportunity of using the extracted patterns as building blocks for more sophisticated analyses.

Going in this direction, we present a series of applications built on top of our multilayer core decomposition. First we focus on the problem of extracting only the *maximal* or, as we call them in this work, the *inner-most cores*, i.e., cores that are not "dominated" by any other core As we will see experimentally, inner-most cores are orders of magnitude less than all the cores. Therefore, it is interesting to develop algorithms that effectively exploit the maximality property and extract inner-most cores directly, without first computing a complete decomposition. Then, we show how multilayer core decomposition finds application to the problem of *densest-subgraph extraction from multilayer networks* [136, 56]. As a further application, we exploit multilayer core decomposition to speed-up the extraction of *frequent cross-graph quasi-cliques* [137]. Finally, we show how multilayer core decomposition can be used to generalize the *community-search* problem [215] to the multilayer setting.

Challenges, contributions, and roadmap

Let $G = (V, E, L)$ be a multilayer graph, where V is a set of vertices, L is a set of layers, and $E \subseteq V \times V \times L$ is a set of edges. Given an $|L|$-dimensional integer vector $\vec{k} = [k_\ell]_{\ell \in L}$, the *multilayer* $\vec{k}$*-core* of G is a maximal subgraph whose vertices have at least degree k_ℓ in that subgraph, for all

[1] Another quite popular terminology adopted in the literature to denote networks with multiple layers and no inter-layer links is "*multiplex networks*".

layers $\ell \in L$ [19]. Vector $\vec{k}$ is dubbed *coreness vector* of that core. The set of all *non-empty* and *distinct* multilayer cores constitutes the *multilayer core decomposition* of G. A major challenge of computing the complete core decomposition of multilayer networks is that *the number of multilayer cores can be exponential in the number of layers*, which makes the problem inherently hard, as the potentially exponential size of the output precludes the existence of polynomial-time algorithms in the general case. In fact, unlike the single-layer case where cores are all nested into each other, no total order exists among multilayer cores. Rather, they form a *core lattice* defining a relation of partial containment. As a result, the multilayer core-decomposition problem cannot be solved in linear time like in single-layer graphs: algorithms in the multilayer setting must be crafted carefully to handle this exponential blowup, and avoid, to the maximum possible extent, the computation of unnecessary (i.e., empty or non-distinct) cores.

A naïve way of computing a multilayer core decomposition consists in generating all possible coreness vectors $\vec{k}$, run for each vector a subroutine that iteratively removes vertices whose degree in some layer ℓ is less than the ℓ-th component of $\vec{k}$, and filter out duplicated cores. This method has evident efficiency issues, as every core is computed starting from the whole input graph, and a significant number of unnecessary (i.e., empty or non-distinct) cores may be generated. Within this view, our first contribution is to devise three algorithms that exploit effective pruning rules during the visit of the lattice, thus being much more efficient than the naïve counterpart. The first two methods are based on a BFS and a DFS strategy, respectively: the BFS method exploits the rule that a core is contained into the intersection of all its fathers in the lattice, while the DFS method iteratively performs a single-layer core decomposition that computes cores along a path from a non-leaf lattice core to a leaf all at once. The third method adopts a HYBRID strategy embracing the main pros of BFS and DFS, and equipped with a *look-ahead* mechanism to skip non-distinct cores.

We then shift the attention to the problem of computing *all and only the inner-most cores*, i.e., the cores that are not dominated by any other core in terms of their index on all the layers. A straightforward way of approaching this problem would be to first compute the complete core decomposition, and then filter out the non-inner-most cores. However, as the inner-most cores are usually much less than the overall cores, it would be desirable to have a method that effectively exploits the maximality property and extracts the inner-most ones directly, without computing a complete decomposition. The design of an algorithm of this kind is an interesting challenge, as it contrasts the intrinsic conceptual properties of core decomposition, based on which a core of order k (in one layer) can be efficiently computed from the core of order $k-1$, of which it is a subset, thus naturally suggesting a bottom-up discovery. For this reason, at first glance, the computation of the core of the highest order would seem as hard as computing the overall core decomposition. In this work we show that, by means of a clever core-lattice visiting strategy, we can prune huge portions of the search space, thus achieving higher efficiency than computing the whole decomposition.

As a major application of multilayer core decomposition, we then focus on the problem of *extracting the densest subgraph from a multilayer network*. Other methods in the literature, i.e., the ones defined in [56, 136, 195, 210], aim at extracting a subgraph that maximizes the minimum average degree over *all* layers. A major limitation of this formulation is that, considering all layers, even the noisy/insignificant layers would contribute to selecting the output subgraph, which would be not really dense, thus preventing us from finding a subgraph being dense in a still large subset of layers. Another simplistic approach at the other end of the spectrum corresponds to flattening the input multilayer graph and resorting to single-layer densest-subgraph extraction. However, this would mean disregarding the different semantics of the layers, incurring in a severe information loss. Within this view, in this work we generalize the problem studied in [56, 136, 195, 210] by introducing a formulation that accounts for a trade-off between high density and number of layers exhibiting the high density. Specifically, given a multilayer graph $G = (V, E, L)$, the average-degree density of a subset of vertices S in a layer ℓ is defined as the number of edges induced by S in ℓ divided by the size of S, i.e., $\frac{|E_\ell[S]|}{|S|}$. We define the *multilayer densest subgraph* as the subset of vertices S^* such that the function

$$\max_{\hat{L} \subseteq L} \min_{\ell \in \hat{L}} \frac{|E_\ell[S^*]|}{|S^*|} |\hat{L}|^\beta \tag{3.1}$$

is maximized. $\beta \in \mathbb{R}^+$ is a parameter controlling the importance of the two sides of the same coin of our problem, i.e., high density and number of layers exhibiting such a density. It can be observed that this problem statement naturally achieves the desired trade-off: the larger the subset $\hat{L}$ of selected layers, the smaller the minimum density $\min_{\ell \in \hat{L}} \frac{|E_\ell[S]|}{|S|}$ registered in those layers. Similarly to the single-layer case in which the core decomposition can be used to obtain a $\frac{1}{2}$-approximation of the densest subgraph, in this work we show that computing the multilayer core decomposition of the input graph and selecting the core maximizing the proposed multilayer density function achieves a $\frac{1}{2|L|^\beta}$-approximation for the general multilayer-densest-subgraph problem formulation, and a $\frac{1}{2}$-approximation for the all-layer specific variant studied in [136, 56].

As a further application of our multilayer core-decomposition tool, we show how it can be used as a profitable preprocessing step to speed-up the problem of *extracting frequent cross-graph quasi-cliques* defined in [137]. Specifically, we prove that the search of frequent cross-graph quasi-cliques can be circumstantiated to a number of restricted areas of the input multilayer graph, corresponding to multilayer cores that comply with the quasi-clique condition. This allows for skipping visiting unnecessary parts of the input graph, and, thus, speeding up the whole process, no matter which specific algorithm is used.

Finally, we also provide a generalization of the *community-search* problem [215] to the multilayer setting, and show how to exploit multilayer core decomposition to obtain optimal solutions to this problem.

Summarizing, this work has the following contributions:

- We define the problem of *core decomposition in multilayer networks*, characterizing its usefulness, its relation to other problems, and its intrinsic complexity. We then devise three algorithms that solve multilayer core decomposition efficiently based on different pruning techniques (Section 3.2).
- We devise further algorithms that are specifically suited for the computation of the *innermost cores* only (Section 3.3).
- We study the problem of *densest-subgraph extraction in multilayer networks*, by devising a novel formulation as an optimization problem that trades-off between high density and number of layers exhibiting high density. We exploit multilayer core decomposition to solve the multilayer densest-subgraph problem with provable approximation guarantees (Section 3.4).
- We show how the multilayer core-decomposition tool can be exploited to speed up the extraction of *frequent cross-graph quasi-cliques* (Section 3.5).
- We formulate the *multilayer community-search problem* and show that multilayer core decomposition provides an optimal solution to this problem (Section 3.6).

An extensive experimental evaluation on a large variety of real multilayer networks is reported in order to assess the effectiveness of the proposed methods in all the aforementioned contexts. For each of these contexts, experiments are provided within the corresponding section.

3.1 Multilayer core decomposition and related problems

In this section we introduce the needed preliminaries and notation, we provide some fundamental properties of multilayer cores, and then formally define all the problems studied in this work.

3.1.1 Multilayer core decomposition

We are given an undirected multilayer graph $G = (V, E, L)$, where V is a set of vertices, L is a set of layers, and $E \subseteq V \times V \times L$ is a set of edges. Let E_ℓ denote the subset of edges in layer $\ell \in L$. For a vertex $u \in V$ we denote by $deg(u, \ell)$ and $deg(u)$ its degree in layer ℓ and over all layers, respectively, i.e., $deg(u, \ell) = |\{e = (u, v, \ell) : e \in E_\ell\}|$, $deg(u) = |\{e = (u, v, \ell) : e \in E\}| = \sum_{\ell \in L} deg(u, \ell)$.

For a subset of vertices $S \subseteq V$ we denote by $G[S]$ the subgraph of G induced by S, i.e., $G[S] = (S, E[S], L)$, where $E[S] = \{e = (u, v, \ell) \mid e \in E, u \in S, v \in S\}$. For a vertex $u \in V$ we denote by $deg_S(u, \ell)$ and $deg_S(u)$ its degree in subgraph S considering layer ℓ only and all layers,

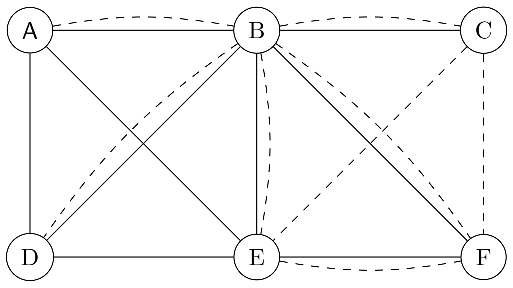


Figure 3.1: Example 2-layer graph (solid edges refer to the first layer, while dashed edges to the second layer) with the following $\vec{k}$-cores: $(0,0) = (1,0) = (0,1) = (1,1) = \{A,B,C,D,E,F\}$, $(2,0) = (2,1) = \{$A,B,D,E,F$\}$, $(3,0) = (3,1) = \{$A,B,D,E$\}$, $(0,2) = (1,2) = (0,3) = (1,3) = \{B,C,E,F\}$, $(2,2) = \{$B,E,F$\}$.

respectively, i.e., $deg_S(u,\ell) = |\{e = (u,v,\ell) : e \in E_\ell[S]\}|$, $deg_S(u) = |\{e = (u,v,\ell) : e \in E[S]\}| = \sum_{\ell \in L} deg_S(u,\ell)$. Finally, let $\mu(\ell)$ and $\mu(\hat{L})$ denote the minimum degree of a vertex in layer ℓ and in a subset $\hat{L} \subseteq L$ of layers, respectively. Let also $\mu(S,\ell)$ and $\mu(S,\hat{L})$ denote the corresponding counterparts of $\mu(\ell)$ and $\mu(\hat{L})$ for a subgraph (induced by a vertex set) S.

A core of a multilayer graph is characterized by an $|L|$-dimensional integer vector $\vec{k} = [k_\ell]_{\ell \in L}$, termed *coreness vector*, whose components k_ℓ denote the minimum degree allowed in layer ℓ. This corresponds to the notion of $\vec{k}$-core introduced by Azimi-Tafreshi et al. [19] for the multilayer core-percolation problem. We recall that Azimi-Tafreshi et al. do not study (or devise any algorithm for) the problem of computing the entire core decomposition of a multilayer graph. Core percolation is studied by analyzing *a single* core of interest computed with the simple iterative-peeling algorithm (Algorithm 3.1). Formally:

Definition 3.1 (multilayer core and coreness vector [19]). *Given a multilayer graph $G = (V,E,L)$ and an $|L|$-dimensional integer vector $\vec{k} = [k_\ell]_{\ell \in L}$, the multilayer $\vec{k}$-core of G is a* maximal *subgraph $G[C] = (C \subseteq V, E[C], L)$ such that $\forall \ell \in L : \mu(C,\ell) \geq k_\ell$. The vector $\vec{k}$ is referred to as the* coreness vector *of $G[C]$.*

Given a coreness vector $\vec{k}$, we denote by $C_{\vec{k}}$ the corresponding core. Also, as a $\vec{k}$-core is fully identified by the vertices belonging to it, we hereinafter refer to it by its vertex set $C_{\vec{k}}$ and the induced subgraph $G[C_{\vec{k}}]$ interchangeably.

It is important noticing that a set of vertices $C \subseteq V$ may correspond to multiple cores. For instance, in the graph in Figure 3.1 the set $\{$A,B,D,E$\}$ corresponds to both $(3,0)$-core and $(3,1)$-core. In other words, a multilayer core can be described by more than one coreness vector. However, as formally shown next, among such multiple coreness vectors there exists one and only one that is not dominated by any other. We call this vector the *maximal coreness vector* of C. In the example in Figure 3.1 the maximal coreness vector of $\{$A,B,D,E$\}$ is $(3,1)$.

Definition 3.2 (maximal coreness vector). *Let $G = (V,E,L)$ be a multilayer graph, $C \subseteq V$ be a core of G, and $\vec{k} = [k_\ell]_{\ell \in L}$ be a coreness vector of C. $\vec{k}$ is said* maximal *if there does not exist any coreness vector $\vec{k'} = [k'_\ell]_{\ell \in L}$ of C such that $\forall \ell \in L : k'_\ell \geq k_\ell$ and $\exists \hat{\ell} \in L : k'_{\hat{\ell}} > k_{\hat{\ell}}$.*

Theorem 3.1. *Multilayer cores have a unique maximal coreness vector.*

Proof. We prove the theorem by contradiction. Assume two maximal coreness vectors $\vec{k} = [k_\ell]_{\ell \in L} \neq \vec{k'} = [k'_\ell]_{\ell \in L}$ exist for a multilayer core C. As $\vec{k} \neq \vec{k'}$ and they are both maximal, there exist two layers $\hat{\ell}$ and $\bar{\ell}$ such that $k_{\hat{\ell}} > k'_{\hat{\ell}}$ and $k'_{\bar{\ell}} > k_{\bar{\ell}}$. By definition of multilayer core (Definition 3.1), it holds that $\forall \ell \in L : \mu(C,\ell) \geq k_\ell, \mu(C,\ell) \geq k'_\ell$. This means that the vector $\vec{k}^* = [k^*_\ell]_{\ell \in L}$, with $k^*_\ell = \max\{k_\ell, k'_\ell\}, \forall \ell \in L$, is a further coreness vector of C. For this vector it holds that $\forall \ell \neq \hat{\ell}, \ell \neq \bar{\ell} : k^*_\ell \geq k'_\ell$, $k^*_{\hat{\ell}} > k'_{\hat{\ell}}$, and $k^*_{\bar{\ell}} > k_{\bar{\ell}}$. Thus, $\vec{k}^*$ dominates both $\vec{k}$ and $\vec{k'}$, which contradicts the hypothesis of maximality of $\vec{k}$ and $\vec{k'}$. The theorem follows. $\square$

The first (and main) problem we tackle in this work is the computation of the complete multilayer core decomposition, i.e., the set of all non-empty multilayer cores.

Problem 3.1 (MULTILAYER CORE DECOMPOSITION). *Given a multilayer graph $G = (V, E, L)$, find the set of all* non-empty *and* distinct *cores of G, along with their corresponding maximal coreness vectors. Such a set forms what we hereinafter refer to as the* multilayer core decomposition *of G.*

3.1.2 Inner-most multilayer cores

Cores of a single-layer graph are all nested one into another. This makes it possible to define the notions of (*i*) *inner-most core*, defined as the core of highest order, and (*ii*) *core index* (or *core number*) of a vertex u, which is the highest order of a core containing u. In the multilayer setting the picture is more complex, as multilayer cores are not necessarily all nested into each other. As a result, the core index of a vertex is not unambiguously defined, while there can exist multiple inner-most cores.

Definition 3.3 (inner-most multilayer cores). *The* inner-most cores *of a multilayer graph are all those cores with maximal coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$ such that there does not exist any other core with coreness vector $\vec{k}' = [k'_\ell]_{\ell \in L}$ where $\forall \ell \in L : k'_\ell \geq k_\ell$ and $\exists \hat{\ell} \in L : k'_{\hat{\ell}} > k_{\hat{\ell}}$.*

To this purpose, look at the example in Figure 3.1. It can be observed that: (*i*) cores are not nested into each other, (*ii*) $(3, 1)$-core, $(1, 3)$-core and $(2, 2)$-core are the inner-most cores, and (*iii*) vertices B and E belong to (inner-most) cores $(3, 1)$, $(1, 3)$, and $(2, 2)$, thus making their core index not unambiguously defined.

The second problem we tackle in this work is the development of smart algorithms to compute all the inner-most cores, without the need of computing the complete multilayer core decomposition.

Problem 3.2 (Inner-most cores computation). *Given a multilayer graph $G = (V, E, L)$, find the set of all* non-empty *and inner-most cores of G, along with their corresponding maximal coreness vectors.*

3.1.3 Multilayer densest subgraph

As anticipated in Section 3, the densest subgraph of a multilayer graph should provide a good trade-off between large density and the number of layers where such a large density is exhibited. We achieve this intuition by means of the following optimization problem:

Problem 3.3 (MULTILAYER DENSEST SUBGRAPH). *Given a multilayer graph $G = (V, E, L)$, a positive real number β, and a real-valued function $\delta : 2^V \to \mathbb{R}^+$ defined as:*

$$\delta(S) = \max_{\hat{L} \subseteq L} \min_{\ell \in \hat{L}} \frac{|E_\ell[S]|}{|S|} |\hat{L}|^\beta, \tag{3.2}$$

find a subset $S^* \subseteq V$ of vertices that maximizes function δ, i.e.,

$$S^* = \arg\max_{S \subseteq V} \delta(S). \tag{3.3}$$

The role of parameter β in Problem 3.3 is to control the importance of the two ingredients of the objective function δ, i.e., density and number of layers exhibiting such a density: the smaller β the larger the importance to be given to the former aspect (density), and vice versa. Also, as a nice side effect, solving the MULTILAYER DENSEST SUBGRAPH problem allows for automatically finding a set of layers of interest for the densest subgraph S^*. In Section 3.4 we will show how to exploit it to devise an algorithm with approximation guarantees for MULTILAYER DENSEST SUBGRAPH, thus extending to the multilayer case the intuition at the basis of the well-known $\frac{1}{2}$-approximation algorithm [18, 55] for single-layer densest-subgraph extraction.

3.1.4 Frequent cross-graph quasi-cliques

Another interesting insight into the notion of multilayer cores is about their relationship with (quasi-)cliques. In single-layer graphs it is well-known that cores can be exploited to speed-up the problem of finding cliques, as a clique of size k is guaranteed to be contained in the $(k-1)$-core. Interestingly, a similar relationship holds in the multilayer context too. Given a multilayer graph $G=(V,E,L)$, a layer $\ell \in L$, and a real number $\gamma \in (0,1]$, a subgraph $G[S]=(S \subseteq V, E[S], L)$ of G is said to be a *γ-quasi-clique* in layer ℓ if all its vertices have at least $\gamma(|S|-1)$ neighbors in layer ℓ within S, i.e., $\forall u \in S : deg_S(u,\ell) \geq \gamma(|S|-1)$. Jiang et al. [137] study the problem of extracting *frequent cross-graph quasi-cliques*, defined next.

Problem 3.4 (Frequent cross-graph quasi-cliques mining [137]). *Given a multilayer graph $G=(V,E,L)$, a function $\Gamma : L \to (0,1]$ assigning a real value to every layer in L, a real number* $\mathrm{min\_sup} \in (0,1]$*, and an integer* $\mathrm{min\_size} \geq 1$*, find all maximal subgraphs $G[S]$ of G of size larger than* min_size *such that there exist at least* $\mathrm{min\_sup} \times |L|$ *layers ℓ for which $G[S]$ is a $\Gamma(\ell)$-quasi-clique.*

In Section 3.5 we will prove that a frequent cross-graph quasi-clique of size K is necessarily contained into a $\vec{k}$-core described by a maximal coreness vector $\vec{k}=[k_\ell]_{\ell \in L}$ such that there exists a fraction of at least *min_sup* layers ℓ where $k_\ell = \lfloor \Gamma(\ell)(K-1) \rfloor$. Based on this property we will show how, by exploiting multilayer core decomposition as a preprocessing step, we can speed-up any algorithm for Problem 3.4.

3.1.5 Multilayer community search

The last application of multilayer core decomposition we study is the so called *community search* problem. Given a graph $G=(V,E)$ and a set $V_Q \subseteq V$ of query vertices, a very wide family of problem requires to find a connected subgraph H of G, which contains all query vertices V_Q and exhibits an adequate degree of cohesiveness, compactness, or density. This type of problem has been termed in the literature in different ways, e.g., *community search* [215, 64, 25], *seed set expansion* [11, 145], *connectivity subgraphs* [83, 226, 202, 6, 201], just to mention a few: see [132] for a recent survey. In this work we adopt the early definition by Sozio and Gionis [215] which measures the cohesiveness of the resulting subgraph by means of the minimum degree inside the subgraph, and we adapt it to the multilayer setting as follows.

Problem 3.5 (Multilayer Community Search). *Given a multilayer graph $G=(V,E,L)$, a set of vertices $S \subseteq V$, and a set of layers $\hat{L} \subseteq L$, we define the minimum degree of a vertex in S, within the subgraph induced by S and $\hat{L}$ as:*

$$\varphi(S,\hat{L}) = \min_{\ell \in \hat{L}} \min_{u \in S} deg_S(u,\ell). \tag{3.4}$$

Given a positive real number β, we define a real-valued density function $\vartheta : 2^V \to \mathbb{R}^+$ as:

$$\vartheta(S) = \max_{\hat{L} \subseteq L} \varphi(S,\hat{L})|\hat{L}|^\beta. \tag{3.5}$$

Given a set $V_Q \subseteq V$ of query vertices, find a subgraph containing all the query vertices and maximizing the density function, i.e.,

$$S^* = \operatorname*{arg\,max}_{V_Q \subseteq S \subseteq V} \vartheta(S). \tag{3.6}$$

In Section 3.6 we will show how to adapt multilayer core decomposition to efficiently provide an exact solution to Problem 3.5.

3.2 Algorithms for multilayer core decomposition

A major challenge of the Multilayer Core Decomposition problem is that the number of multilayer cores to be output may be exponential in the number of layers. Specifically, denoting by

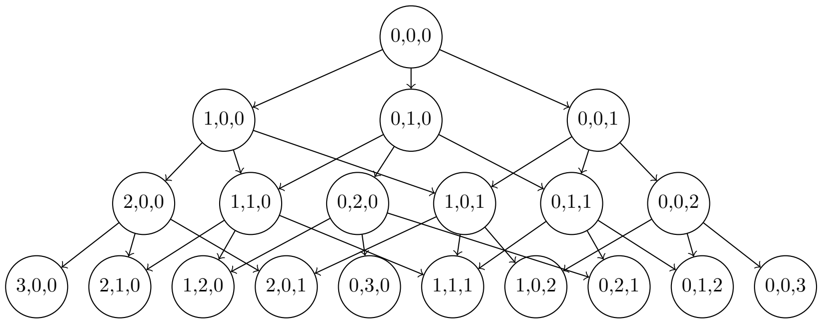


Figure 3.2: Section of core lattice of a 3-layer graph.

K_ℓ the maximum order of a core for layer ℓ, the number of multilayer cores is $\mathcal{O}(\prod_{\ell \in L} K_\ell)$. This makes Multilayer Core Decomposition intrinsically hard: *in the general case, no polynomial-time algorithm can exist.* The challenge in this context hence lies in handling this exponential blowup by early recognizing and skipping unnecessary portions of the search space, such as non-distinct and/or empty cores.

Given a multilayer graph $G = (V, E, L)$ and a coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$, finding the corresponding core can easily be solved in $\mathcal{O}(|E| + |V| \times |L|)$ time by iteratively removing a vertex u having $deg_{G'}(u, \ell) < k_\ell$ in some layer ℓ, where G' denotes the current graph resulting from all previous vertex removals (Algorithm 3.1, where the set S of vertices to be considered is set to $S = V$). Therefore, a naïve algorithm to compute the entire multilayer core decomposition consists of generating all possible coreness vectors, run the multilayer core-detection algorithm just described for each of such vectors, and retain only non-empty and distinct cores. This naïve method requires all vectors $[k_\ell]_{\ell \in L}$, where each k_ℓ component is varied within the interval $[0..K_\ell]$.[2] This corresponds to a $\Theta(\prod_{\ell \in L} K_\ell)$ number of vectors. As a result, the overall time complexity of the method is $\mathcal{O}\big((|E| + |V| \times |L|) \times \prod_{\ell \in L} K_\ell\big)$.

This approach has two major weaknesses: (*i*) each core is computed starting from the whole input graph, and (*ii*) by enumerating all possible coreness vectors beforehand a lot of non-distinct and/or empty (thus, unnecessary) cores may be computed. In the following we present three methods that solve Multilayer Core Decomposition much more efficiently.

3.2.1 Search space

Although multilayer cores are not all nested into each other, a notion of partial containment can still be defined. Indeed, it can easily be observed that a $\vec{k}$-core with coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$ is contained into any $\vec{k}'$-core described by a coreness vector $\vec{k}' = [k'_\ell]_{\ell \in L}$ whose components k'_ℓ are all no more than components k_ℓ, i.e., $k'_\ell \leq k_\ell$, $\forall \ell \in L$. This result is formalized next:

Fact 3.1. *Given a multilayer graph $G = (V, E, L)$ and two cores $C_{\vec{k}}$ and $C_{\vec{k}'}$ of G with coreness vectors $\vec{k} = [k_\ell]_{\ell \in L}$ and $\vec{k}' = [k'_\ell]_{\ell \in L}$, respectively, it holds that if $\forall \ell \in L : k'_\ell \leq k_\ell$, then $C_{\vec{k}} \subseteq C_{\vec{k}'}$.*

Proof. Combining the definition of multilayer core (Definition 3.1) and the hypothesis on vectors $\vec{k}$ and $\vec{k}'$, it holds that $\forall \ell \in L : \mu(C_{\vec{k}}, \ell) \geq k_\ell \geq k'_\ell$. This means that $C_{\vec{k}}$ satisfies the definition of $\vec{k}'$-core, thus implying that all vertices in $C_{\vec{k}}$ are part of $C_{\vec{k}'}$ too. The fact follows. □

Based on Fact 3.1, the search space of our problem can be represented as a lattice defining a partial order among all cores (Figure 3.2). Such a lattice, which we call the *core lattice*, corresponds to a DAG where nodes represent cores,[3] and links represent relationships of containment between cores (a "father" node contains all its "child" nodes). We assume the core lattice keeping track of non-empty and not necessarily distinct cores: a core is present in the lattice as many times as

[2] K_ℓ values can be derived beforehand by computing a single-layer core decomposition in each layer ℓ. This process overall takes $\mathcal{O}(|E|)$ time.

[3] Throughout the chapter we use the term "node" to refer to elements of the core lattice, and "vertex" for the elements of the multilayer graph.

Algorithm 3.1: $\vec{k}$-core

```
Input: A multilayer graph $G = (V, E, L)$, a set $S \subseteq V$ of vertices, an $|L|$-dimensional
       integer vector $\vec{k} = [k_\ell]_{\ell \in L}$.
Output: The $\vec{k}$-core $C_{\vec{k}}$ of $G$.
1 while $\exists u \in S, \exists \ell \in L : deg_S(u, \ell) < k_\ell$ do
2     $S \leftarrow S \setminus \{u\}$
3 $C_{\vec{k}} = S$
```

Algorithm 3.2: BFS-ML-cores

```
Input: A multilayer graph $G = (V, E, L)$.
Output: The set $\mathbf{C}$ of all non-empty multilayer cores of $G$.
1  $\mathbf{C} \leftarrow \emptyset$, $\mathbf{Q} \leftarrow \{[0]_{|L|}\}$, $\mathcal{F}([0]_{|L|}) \leftarrow \emptyset$        // $\mathcal{F}$ keeps track of father nodes
2  while $\mathbf{Q} \neq \emptyset$ do
3      dequeue $\vec{k} = [k_\ell]_{\ell \in L}$ from $\mathbf{Q}$
4      if $|\{k_\ell : k_\ell > 0\}| = |\mathcal{F}(\vec{k})|$ then                          // Corollary 3.2
5          $F_\cap \leftarrow \bigcap_{F \in \mathcal{F}(\vec{k})} F$                                 // Corollary 3.1
6          $C_{\vec{k}} \leftarrow \vec{k}$-core$(G, F_\cap, \vec{k})$                              // Algorithm 3.1
7          if $C_{\vec{k}} \neq \emptyset$ then
8              $\mathbf{C} \leftarrow \mathbf{C} \cup \{C_{\vec{k}}\}$
9              forall $\ell \in L$ do                                      // enqueue child nodes
10                 $\vec{k}' \leftarrow [k_1, \ldots, k_\ell + 1, \ldots, k_{|L|}]$
11                 enqueue $\vec{k}'$ into $\mathbf{Q}$
12                 $\mathcal{F}(\vec{k}') \leftarrow \mathcal{F}(\vec{k}') \cup \{C_{\vec{k}}\}$
```

the number of its coreness vectors. Each level i of the lattice represents the children of cores at lattice level $i-1$. In particular, level i contains all those cores whose coreness vector results from increasing one and only one component of its fathers' coreness vector by one. Formally, a lattice level i contains all $\vec{k}$-cores with coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$ such that there exists a core at lattice level $i-1$ with coreness vector $\vec{k}' = [k'_\ell]_{\ell \in L}$ where: $\exists \ell \in L : k_\ell = k'_\ell + 1$, and $\forall \hat{\ell} \neq \ell : k_{\hat{\ell}} = k'_{\hat{\ell}}$. As a result, level 0 contains the root only, which corresponds to the whole input graph (i.e., the $[0]_{|L|}$-core), the leaves correspond to inner-most cores, and any non-leaf node has at least one and at most $|L|$ children. Moreover, every level i contains all cores whose coreness-vector components sum to i.

Solving the Multilayer Core Decomposition problem is hence equivalent to building the core lattice of the input graph. The efficient methods we present next are all based on smart core-lattice building strategies that extract cores from smaller subgraphs, while also attempting to minimize the visit/computation of unnecessary (i.e., empty/non-distinct) cores.

3.2.2 Breadth-first algorithm

Two interesting corollaries can be derived from Fact 3.1. First, any non-empty $\vec{k}$-core is necessarily contained in the intersection of all its father nodes of the core lattice. Second, any non-empty $\vec{k}$-core has *exactly* as many fathers as the number of non-zero components of its coreness vector $\vec{k}$:

Corollary 3.1. *Given a multilayer graph G, let C be a core of G and $\mathcal{F}(C)$ be the set of fathers of C in the core lattice of G. It holds that $C \subseteq \bigcap_{\hat{C} \in \mathcal{F}(C)} \hat{C}$.*

Proof. By definition of core lattice, the coreness vector of all father cores $\mathcal{F}(C)$ of C is dominated by the coreness vector of C. Thus, according to Fact 3.1, it holds that $C \subseteq C'$, $\forall C' \in \mathcal{F}(C)$. Assume a vertex $u \in C$, $u \notin \bigcap_{\hat{C} \in \mathcal{F}(C)} \hat{C}$ exists. This implies that there exists a father core $C' \in \mathcal{F}(C)$ such that $C \nsubseteq C'$, thus leading to a contradiction. □

Algorithm 3.3: DFS-ML-cores

```
Input: A multilayer graph $G = (V, E, L)$.
Output: The set $\mathbf{C}$ of all non-empty multilayer cores of $G$.
1 $\mathbf{C} \leftarrow \{V\}, \quad R \leftarrow L, \quad \mathbf{Q} \leftarrow \{[0]_{|L|}\}, \quad \mathbf{Q}' \leftarrow \emptyset$
2 while $R \neq \emptyset$ do
3     remove a layer from $R$1; forall $\vec{k} \in \mathbf{Q}$ do
4         $\forall \ell \in R$ s.t. $k_\ell = 0 : \mathbf{Q}' \leftarrow \mathbf{Q}' \cup \{\vec{k}' \mid C_{\vec{k}'} \in \vec{k}\text{-coresPath}(G, C_{\vec{k}}, \vec{k}, \ell)\}$
5         $\forall \ell \in L \setminus R$ s.t. $k_\ell = 0 : \mathbf{C} \leftarrow \mathbf{C} \cup \vec{k}\text{-coresPath}(G, C_{\vec{k}}, \vec{k}, \ell)$
6     $\mathbf{C} \leftarrow \mathbf{C} \cup \{C_{\vec{k}} \mid \vec{k} \in \mathbf{Q}'\}, \quad \mathbf{Q} \leftarrow \mathbf{Q}', \quad \mathbf{Q}' \leftarrow \emptyset$
```

Corollary 3.2. *Given a multilayer graph G, let C be a core of G with coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$, and $\mathcal{F}(C)$ be the set of fathers of C in the core lattice of G. It holds that $|\mathcal{F}(C)| = |\{k_\ell : \ell \in L, k_\ell > 0\}|$.*

Proof. By definition of core lattice, a core C at level i is assigned a coreness vector whose components sum to i, while the fathers $\mathcal{F}(C)$ of C have coreness vector whose components sum to $i - 1$. Then, the coreness vector of a father of C can be obtained by decreasing a non-zero component of the coreness vector of C by one (zero components would lead to negative coreness vector components, thus they do not count). This means that the number of fathers of C is upper-bounded by the non-zero components of its coreness vector. More precisely, the number of fathers of C is exactly equal to this number, as, according to Corollary 3.1, no father of C can be empty, otherwise C would be empty too and would not be part of the core lattice. □

The above corollaries pave the way to a breadth-first search building strategy of the core lattice, where cores are generated level-by-level by properly exploiting the rules in the two corollaries (Algorithm 3.2). Although the worst-case time complexity of this BFS-ML-cores method remains unchanged with respect to the naïve algorithm, the BFS method is expected to be much more efficient in practice, due to the following main features: (*i*) cores are not computed from the initial graph every time, but from a much smaller subgraph given by the intersection of all their fathers; (*ii*) in many cases, i.e., when the rule in Corollary 2 (which can be checked in constant time) arises, no overhead due to the intersection among father cores is required; (*iii*) the number of empty cores computed is limited, as no empty core may be generated from a core that has already been recognized as empty.

3.2.3 Depth-first algorithm

Although being much smarter than the naïve method, BFS-ML-cores still has some limitations. First, it visits every core as many times as the number of its fathers in the core lattice. Also, as a second limitation, consider a path $\mathcal{P}$ of the lattice connecting a non-leaf node to a leaf by varying the same ℓ-th component of the corresponding coreness vectors. It is easy to see that the computation of all cores within $\mathcal{P}$ with BFS-ML-cores takes $\mathcal{O}(|\mathcal{P}| \times (|E| + |V| \times |L|))$ time, as the core-decomposition process is re-started at every level of the lattice. This process can in principle be performed more efficiently, i.e., so as to take $\mathcal{O}(|\mathcal{P}| + |E| + |V| \times |L|)$ time, as it actually corresponds to (a simple variant of) a single-layer core decomposition.

To address the two above cons, we propose a method performing a depth-first search on the core lattice. The method, dubbed DFS-ML-cores (Algorithm 3.3), iteratively picks a non-leaf core $\vec{k} = [k_1, \ldots, k_\ell, \ldots, k_{|L|}]$ and a layer ℓ such that $k_\ell = 0$, and computes all cores $[k_1, \ldots, k_\ell + 1, \ldots, k_{|L|}], \ldots, [k_1, \ldots, K_\ell, \ldots, k_{|L|}]$ with a run of the $\vec{k}$-coresPath$(G, C_{\vec{k}}, \vec{k}, \ell)$ subroutine. Specifically, such a subroutine returns the cores corresponding to all coreness vectors obtained by varying the ℓ-th component of $\vec{k}$ within $[0, \ldots, K_\ell]$. Also, it discards vertices violating the coreness condition specified by vector $\vec{k}$, i.e., vertices whose degree in some layer $\hat{\ell} \neq \ell$ is less than the $\hat{\ell}$-th component of $\vec{k}$. The pseudocode of $\vec{k}$-coresPath is reported as Algorithm 3.4: it closely resembles the traditional core-decomposition algorithm for single-layer graphs, except for

Algorithm 3.4: $\vec{k}$-coresPath

```
Input: A multilayer graph $G = (V, E, L)$, a set $S \subseteq V$ of vertices, an $|L|$-dimensional
       integer vector $\vec{k} = [k_\ell]_{\ell \in L}$, and a layer $\ell \in L$.
Output: The set $\mathbf{C}_{\vec{k},\ell}$ of the multilayer cores of $G$ varying the $\ell$-th component of $\vec{k}$.
1  $\mathbf{C}_{\vec{k},\ell} \leftarrow \emptyset$, $\mathcal{D} \leftarrow \emptyset$
2  forall $u \in S$ do
3      $\mathcal{D}(deg_S(u,\ell)) \leftarrow \mathcal{D}(deg_S(u,\ell)) \cup \{u\}$
4  forall $k \in [0, \ldots, K_\ell]$ do
5      while $D(k) \neq \emptyset$ do
6          remove a vertex $u$ from $\mathcal{D}(k)$
7          $S \leftarrow S \setminus \{u\}$
8          forall $v \in S : (u, v, \ell) \in E \wedge deg_S(v, \ell) \geq k$ do
9              $\mathcal{D}(deg_S(v,\ell) + 1) \leftarrow \mathcal{D}(deg_S(v,\ell) + 1) \setminus \{v\}$
10             $\mathcal{D}(deg_S(v,\ell)) \leftarrow \mathcal{D}(deg_S(v,\ell)) \cup \{v\}$
11         forall $\hat{\ell} \in L \setminus \{\ell\}$ do
12             forall $v \in S : (u, v, \hat{\ell}) \in E \wedge deg_S(v, \hat{\ell}) < k_{\hat{\ell}}$ do
13                 $\mathcal{D}(deg_S(v,\ell)) \leftarrow \mathcal{D}(deg_S(v,\ell)) \setminus \{v\}$
14                 $\mathcal{D}(k) \leftarrow \mathcal{D}(k) \setminus \{v\}$
15     $\mathbf{C}_{\vec{k},\ell} \leftarrow \mathbf{C}_{\vec{k},\ell} \cup \{S\}$
```

the addition of the cycle starting at Line 11, which identifies the aforementioned vertices to be discarded.

A side effect of this strategy is that the same core may be computed multiple times. As an example, in Figure 3.2 the $(1, 2, 0)$-core is computed by core decompositions initiated at both cores $(1, 0, 0)$ and $(0, 2, 0)$. To reduce (but not eliminate) these multiple core computations, the DFS-ML-cores method exploits the following result.

Theorem 3.2. *Given a multilayer graph $G = (V, E, L)$, let $[\ell_1, \ldots, \ell_{|L|}]$ be an order defined over set L. Let $\mathbf{Q}_0 = \{[0]_{|L|}\}$, and, $\forall i \in [1..|L|]$, let $\mathbf{Q}_i = \{\vec{k}' \mid C_{\vec{k}'} \in \vec{k}\text{-coresPath}(G, C_{\vec{k}}, \vec{k}, \ell), \vec{k} \in \mathbf{Q}_{i-1}, \ell \in (\ell_i..\ell_{|L|}], k_\ell = 0\}$ and $\mathbf{C}_i = \{\vec{k}' \mid C_{\vec{k}'} \in \vec{k}\text{-coresPath}(G, C_{\vec{k}}, \vec{k}, \ell), \vec{k} \in \mathbf{Q}_{i-1}, \ell \in [\ell_1..\ell_i], k_\ell = 0\}$. The set $\mathbf{C} = \{C_{\vec{k}} \mid \vec{k} \in \bigcup_{i=0}^{|L|} \mathbf{Q}_i \cup \bigcup_{i=1}^{|L|} \mathbf{C}_i\}$ is the multilayer core decomposition of G.*

Proof. The multilayer core decomposition of G is formed by the union of all non-empty and distinct cores of all paths $\mathcal{P}$ of the lattice connecting a non-leaf node to a leaf by varying the same ℓ-th component of the corresponding coreness vectors.

For any $i \in [1..|L|]$, let $\mathcal{P}_i \in \mathcal{P}$ denote the subset of paths whose coreness vectors $\vec{k}' = [k'_\ell]_{\ell \in L}$ have a number of non-zero components equal to i. By definition of $\mathbf{Q}_i$ and $\mathbf{C}_i$ it holds that all coreness vectors $\vec{k}'$ of the cores along the paths in $\mathcal{P}_i$ are in $\mathbf{Q}_i \cup \mathbf{C}_i = \{\vec{k}' : |\{k'_\ell : \ell \in L, k'_\ell > 0\}| = i\}$. Also, since some of the paths may overlap, all cores along the paths $\mathcal{P}_i$ are computed by executing single-layer core decompositions initiated at a subset of cores along the paths $\mathcal{P}_{i-1}$. Such a subset of cores is represented by the subset of coreness vectors within $\mathbf{Q}_{i-1} = \{\vec{k} : |\{k_\ell : \ell \in [\ell_2..\ell_{|L|}], k_\ell > 0\}| = i - 1\}$. Moreover, note that single-layer core decompositions for the layers where $k_\ell \neq 0$ are discarded, as it boils down to visit cores in $\mathcal{P}_{i-1}$. As a result, the set $\{C_{\vec{k}} \mid \vec{k} \in \bigcup_{i=0}^{|L|} \mathbf{Q}_i \cup \bigcup_{i=1}^{|L|} \mathbf{C}_i\}$ correctly contains all possible coreness vectors of the core lattice. □

Referring to the pseudocode in Algorithm 3.3, the result in Theorem 3.2 is implemented by keeping track of a subset of layers $R \subseteq L$. At the beginning $R = L$, and, at each iteration of the main cycle, a layer ℓ is removed from it. The output of the algorithm is independent of the layer-removal order, i.e., the algorithm is guaranteed to be sound and complete regardless of the layer ordering. Instead, layer-removal order may in principle affect running time. In our experiments we tested several orders: random, non-decreasing average-degree density, non-increasing average-degree density. All those orders gave comparable results in terms of running time, we therefore

Algorithm 3.5: HYBRID-ML-cores

```
Input: A multilayer graph $G = (V, E, L)$.
Output: The set $\mathbf{C}$ of all non-empty multilayer cores of $G$.
1  $\mathbf{Q} \leftarrow \{[0]_{|L|}\}$, $\mathcal{F}([0]_{|L|}) \leftarrow \emptyset$                // $\mathcal{F}$ keeps track of father nodes
2  $\mathbf{Q}' \leftarrow \bigcup_{\ell \in L}\{\vec{k} \mid C_{\vec{k}} \in \vec{k}\text{-coresPath}(G, V, [0]_{|L|}, \ell)\}$     // looked-ahead cores
3  $\mathbf{C} \leftarrow \{C_{\vec{k}} \mid \vec{k} \in \mathbf{Q}'\}$
4  while $\mathbf{Q} \neq \emptyset$ do
5      dequeue $\vec{k} = [k_\ell]_{\ell \in L}$ from $\mathbf{Q}$
6      if $|\{k_\ell : k_\ell > 0\}| = |\mathcal{F}(\vec{k})| \wedge \vec{k} \notin \mathbf{Q}'$ then    // Corollary 3.2
7          $F_\cap \leftarrow \bigcap_{F \in \mathcal{F}(\vec{k})} F$                       // Corollary 3.1
8          $C_{\vec{k}} \leftarrow \vec{k}\text{-core}(G, F_\cap, \vec{k})$                 // Algorithm 3.1
9          if $C_{\vec{k}} \neq \emptyset$ then
10             $\mathbf{C} \leftarrow \mathbf{C} \cup \{C_{\vec{k}}\}$
11             $\vec{d}_\mu(C_{\vec{k}}) \leftarrow [\mu(C_{\vec{k}}, \ell)]_{\ell \in L}$      // look-ahead mechanism (Corollary 3.3)
12             $\mathbf{Q}' \leftarrow \mathbf{Q}' \cup \{\vec{k}' \mid \vec{k} \leq \vec{k}' \leq \vec{d}_\mu(C_{\vec{k}})\}$
13     if $\vec{k} \in \mathbf{Q}'$ then
14         forall $\ell \in L$ do                                  // enqueue child nodes
15             $\vec{k}' \leftarrow [k_1, \ldots, k_\ell + 1, \ldots, k_{|L|}]$
16             enqueue $\vec{k}$' into $\mathbf{Q}$
17             $\mathcal{F}(\vec{k}') \leftarrow \mathcal{F}(\vec{k}') \cup \{C_{\vec{k}}\}$
```

decided to stick to the the simplest one, i.e., random, in our implementation. Set $\mathbf{Q}$ keeps track of (the coreness vector of) all lattice nodes where the current single-layer core-decomposition processes need to be run from. $\mathbf{Q}'$ stores the (coreness vector of) cores computed from each node in $\mathbf{Q}$ and for each layer within R, while also forming the basis of $\mathbf{Q}$ for the next iteration.

In summary, compared to BFS-ML-cores, the DFS method reduces both the time complexity of computing all cores in a path $\mathcal{P}$ from a non-leaf node to a leaf of the core lattice (from $\mathcal{O}(|\mathcal{P}| \times (|E| + |V| \times |L|))$ to $\mathcal{O}(|\mathcal{P}| + |E| + |V| \times |L|)$), and the number of times a core is *visited*, which may now be smaller than the number of its fathers. On the other hand, DFS-ML-cores comes with the aforementioned issue that some cores may be *computed* multiple times (while in BFS-ML-cores every core is computed only once). Furthermore, cores are computed starting from larger subgraphs, as intersection among multiple fathers can not exploited.

3.2.4 Hybrid algorithm

The ultimate output of both BFS-ML-cores and DFS-ML-cores correctly corresponds to all distinct cores of the input graph and the corresponding maximal coreness vectors.[4] Nevertheless, none of these methods is able to skip the computation of non-distinct cores. Indeed, both methods need to compute every core C as many times as the number of its coreness vectors in order to guarantee completeness. To address this limitation we devise a further method where the main peculiarities of both BFS-ML-cores and DFS-ML-cores are joined into a "hybrid" lattice-visit strategy. This HYBRID-ML-cores method exploits the following corollary of Theorem 3.1, stating that the maximal coreness vector of a core C is given by the vector containing the minimum degree of a vertex in C for each layer:

Corollary 3.3. *Given a multilayer graph $G = (V, E, L)$, the maximal coreness vector of a multilayer core C of G corresponds to the $|L|$-dimensional integer vector $\vec{d}_\mu(C) = [\mu(C, \ell)]_{\ell \in L}$.*

[4]Pseudocodes in Algorithms 3.2 and 3.3 guarantee this as cores are added to a set $\mathbf{C}$ that does not allow duplicates. Any real implementation can easily take care of this by checking whether a core is already in $\mathbf{C}$, and update it in case the corresponding coreness vector contains the previously-stored one.

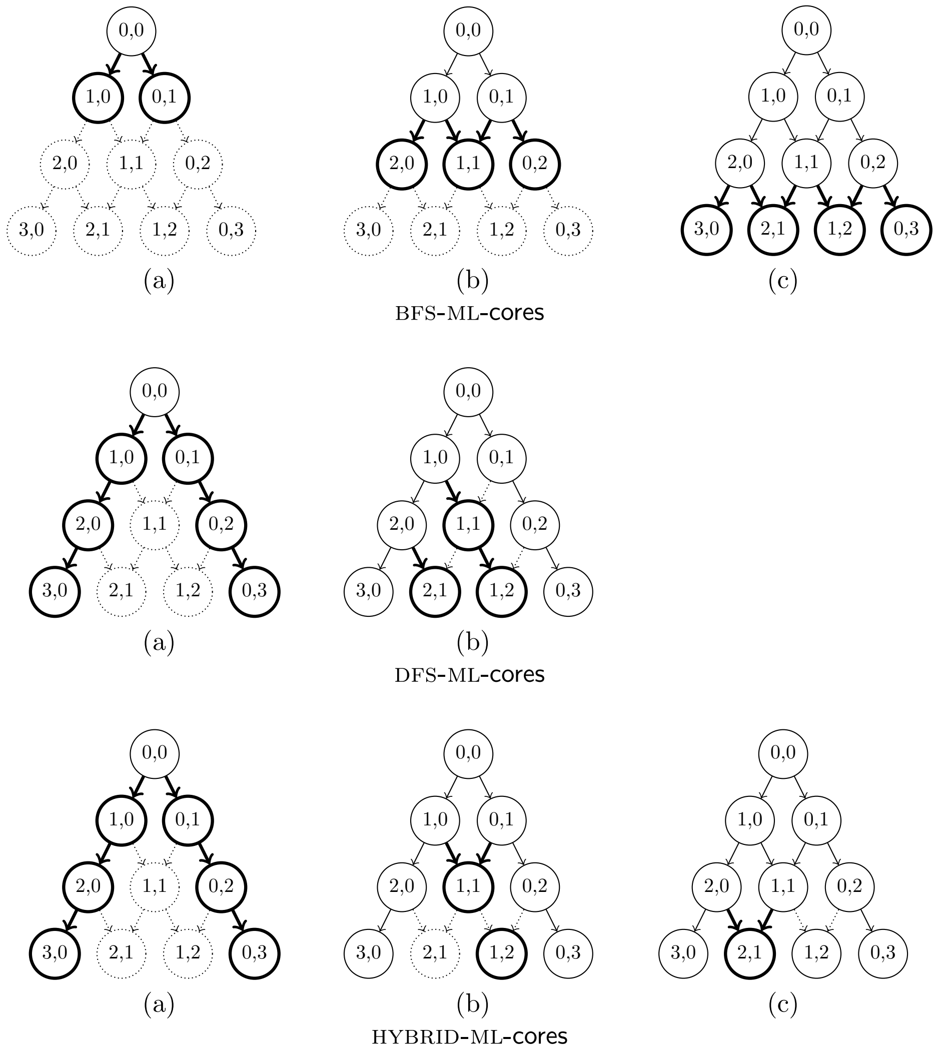


Figure 3.3: Running example of our algorithms for multilayer core decomposition over a core lattice of a 2-layer graph. Nodes and links depicted by solid lines have been visited in previous steps of the algorithm, those in thick lines are visited during the current step, while the remaining in dotted lines have not been visited yet.

Proof. By Definition 3.1, vector $\vec{d_\mu}(C)$ is a coreness vector of C. Assume that $\vec{d_\mu}(C)$ is not maximal, meaning that another coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$ dominating $\vec{d_\mu}(C)$ exists. This implies that $k_\ell \geq \mu(C, \ell)$, and $\exists \hat{\ell} \in L : k_{\hat{\ell}} > \mu(C, \hat{\ell})$. By definition of multilayer core, all vertices in C have degree larger than the minimum degree $\mu(C, \hat{\ell})$ in layer $\hat{\ell}$, which is a clear contradiction. □

Corollary 3.3 gives a rule to skip the computation of non-distinct cores: given a core C with coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$, all cores with coreness vector $\vec{k'} = [k'_\ell]_{\ell \in L}$ such that $\forall \ell \in L : k_\ell \leq k'_\ell \leq \mu(C, \ell)$ are guaranteed to be equal to C and do not need to be explicitly computed. For instance, in Figure 3.2, assume that the min-degree vector of the $(0, 0, 1)$-core is $(0, 1, 2)$. Then, cores $(0, 0, 2)$, $(0, 1, 1)$, and $(0, 1, 2)$ can immediately be set equal to the $(0, 0, 1)$-core. The HYBRID-ML-cores algorithm we present here (Algorithm 3.5) exploits this rule by performing a breadth-first search equipped with a "look-ahead" mechanism resembling a depth-first search. Moreover, HYBRID-ML-cores starts with a single-layer core decomposition for each layer so as to have more fathers early-on for intersections. Cores interested by the look-ahead rule are still *visited* and stored in $\mathbf{Q'}$, as they may be needed for future core computations. However, no further computational overhead is required for them. The time complexity of HYBRID-ML-cores is the same as BFS-ML-cores, plus an additional $\mathcal{O}(|E|)$ time for every visited multilayer core, which is needed in the look-ahead rule to compute the min-degree vector of that core.

3.2.5 Discussion

To have a concrete comparison of the characteristics of the proposed algorithms for multilayer core decomposition, we report in Figure 3.3 a running example over a core lattice of a simple 2-layer graph. All the algorithms start by visiting the root of the core lattice, which corresponds to the whole input multilayer graph (this preliminary step is left out from Figure 3.3 since it is shared by all the methods). BFS-ML-cores visits the core lattice level by level, and exploits every containment relationship. The execution pattern of DFS-ML-cores is instead much different: it starts by finding those multilayer cores having a single component of the coreness vector other than zero and, in a later step, visits the rest of the core lattice. In both steps (a) and (b) DFS-ML-cores visits cores following straight paths in the search space, i.e., from a core to a leaf. As a result, not all the containment relationships are exploited. For instance, the computation of the $(2, 1)$-core exploits the containment from the $(2, 0)$-core, but not from the $(1, 1)$-core. HYBRID-ML-cores is, as expected, a mix of the two other methods. The first step is identical to DFS-ML-cores. At step (b), HYBRID-ML-cores starts to visit the remaining cores by a breadth-first-search strategy, while also exploiting the look-ahead mechanism. In particular, the minimum degree vector of the $(1, 1)$-core is found to be equal to $(1, 2)$; therefore, the $(1, 2)$-core is not computed directly, but set equal to the $(1, 1)$-core. In the final step HYBRID-ML-cores visits the remaining core by going on with the breadth-first search.

We already discussed (in the respective paragraphs) the strengths and weaknesses of BFS-ML-cores and DFS-ML-cores: the best among the two is determined by the peculiarities of the specific input graph. On the other hand, HYBRID-ML-cores profitably exploits the main nice features of both BFS-ML-cores and DFS-ML-cores, thus is expected to outperform both methods in most cases. However, in those graphs where the number of non-distinct cores is limited, the overhead due to the look-ahead mechanism can make the performance of HYBRID-ML-cores degrade.

In terms of space requirements, BFS-ML-cores needs to keep in memory all those cores having at least a child in the queue, i.e., at most two levels of the lattice (the space taken by a multilayer core is $\mathcal{O}(|V|)$). The same applies to HYBRID-ML-cores with the addition of the cores computed through single-layer core decomposition and look-ahead, until all their children have been processed. DFS-ML-cores instead requires to store all cores where the single-layer core-decomposition process should be started from, both in the current iteration and the next one. Thus, we expect DFS-ML-cores to take more space than BFS-ML-cores and HYBRID-ML-cores, as in practice the number of cores to be stored should be more than the cores belonging to two lattice levels.

3.2.6 Experimental results

In this subsection we present experiments to (*i*) compare the proposed algorithms in terms of runtime, memory consumption, and search-space exploration; (*ii*) characterize the output core decompositions, also by comparing total number of cores and number of inner-most cores.

Datasets. We select publicly-available real-world multilayer networks, whose main characteristics are summarized in Table 3.1.

Homo[5] and SacchCere[5] are networks describing different types of genetic interactions between genes in Homo Sapiens and Saccharomyces Cerevisiae, respectively. ObamaInIsrael[5] represents different types of social interaction (e.g., *re-tweeting*, *mentioning*, and *replying*) among Twitter users, focusing on Barack Obama's visit to Israel in 2013. Similarly, Higgs[5] is built by tracking the spread of news about the discovery of the Higgs boson on Twitter, with the additional layer for the *following* relation. Friendfeed[6] contains public interactions among users of Friendfeed collected over two months (e.g., *commenting*, *liking*, and *following*). FriendfeedTwitter[6] is a multi-platform social network, where layers represent interactions within Friendfeed and Twitter between users registered to both platforms [70]. Amazon[7] is a co-purchasing *temporal network*, containing four

[5]https://comunelab.fbk.eu/data.php
[6]http://multilayer.it.uu.se/datasets.html
[7]https://snap.stanford.edu/data/

Table 3.1: Characteristics of the real-world datasets: number of vertices ($|V|$), number of overall edges ($|E|$), number of layers ($|L|$), minimum, average, and maximum number of edges in a layer (min $|E_\ell|$, avg $|E_\ell|$, max $|E_\ell|$), and application domain.

dataset	$\|V\|$	$\|E\|$	$\|L\|$	min $\|E_\ell\|$	avg $\|E_\ell\|$	max $\|E_\ell\|$	domain
Homo	18k	153k	7	256	21k	83k	genetic
SacchCere	6.5k	247k	7	1.3k	35k	91k	genetic
DBLP	513k	1.0M	10	96k	101k	113k	co-authorship
ObamaInIsrael	2.2M	3.8M	3	557k	1.2M	1.8M	social
Amazon	410k	8.1M	4	899k	2.0M	2.4M	co-purchasing
FriendfeedTwitter	155k	13M	2	5.2M	6.8M	8.3M	social
Higgs	456k	13M	4	28k	3.4M	12M	social
Friendfeed	510k	18M	3	226k	6.2M	18M	social

snapshots between March and June 2003. Finally, DBLP[8] is derived following the methodology in [46]. For each co-authorship relation (edge), the bag of words resulting from the titles of all papers co-authored by the two authors is collected. Then *LDA* topic modeling [40] is applied to automatically identify a hundred topics. Among these, ten topics that are recognized as the most relevant to the data-mining area have been hand-picked. Every selected topic corresponds to a layer. An edge between two co-authors in a certain layer exists if the relation between those co-authors is labeled with the topic corresponding to that layer.

Implementation. All methods are implemented in Python (v. 2.7.12) and compiled by Cython: all our code is available at github.com/egalimberti/multilayer_core_decomposition. All experiments are run on a machine equipped with Intel Xeon CPU at 2.1GHz and 128GB RAM except for Figure 3.4, whose results are obtained on Intel Xeon CPU at 2.7GHz with 128GB RAM.

Comparative evaluation. We compare the naïve baseline (for short N) and the three proposed methods BFS-ML-cores (for short BFS), DFS-ML-cores (DFS), HYBRID-ML-cores (H) in terms of running time, memory usage, and number of computed cores (as a measure of the explored search-space portion). The results of this comparison are shown in Table 3.2. As expected, N is the least efficient method: it is outperformed by our algorithms by 1–4 orders of magnitude. Due to its excessive requirements, we could not run it in reasonable time (i.e., 30 days) on the Friendfeed dataset.

Among the proposed methods, H is recognized as the best method (in absolute or with performance comparable to the best one) in the first five (out of a total of eight) datasets. In the remaining three datasets the best method is DFS. This is mainly motivated by the fact that those three datasets have a relatively small number of layers, an aspect which DFS takes particular advantage from (as also better testified by the experiment with varying the number of layers discussed below). In some cases H is also comparable to BFS, thus confirming the fact that in datasets where the number of non-distinct cores is not so large the performance of the two methods gets closer. A similar reasoning holds between BFS and DFS (at least with a small/moderate number of the layers, see next): BFS is faster in most cases, but, due to the respective pros and cons discussed in Section 3.2, it is not surprising that the two methods achieve comparable performance in a number of other cases.

To test the behavior with varying the number of layers, Figure 3.4 shows the running times of the proposed methods on different versions of the DBLP dataset, obtained by selecting a variable number of layers, from 2 to 10. While the performance of the three methods is comparable up to six layers, beyond this threshold the execution time of DFS grows much faster than BFS and H. This attests that the pruning rules of BFS and H are more effective as the layers increase. To summarize, DFS is expected to have runtime comparable to (or better than) BFS and H when the number of layers is small, while H is faster than BFS when the number of non-distinct cores is large.

The number of computed cores is always larger than the output cores as all methods might compute empty cores or, in the case of DFS, the same core multiple times. Table 3.2 shows that DFS computes more cores than BFS and H, which conforms to its design principles.

Finally, all methods turn out to be memory-efficient, taking no more than 1.5GB of memory.

[8] `http://dblp.uni-trier.de/xml/`

Table 3.2: Comparative evaluation: proposed methods and baseline. Runtime differs from [96] since a different server was employed.

dataset	#output cores	method	runtime (s)	memory (MB)	#computed cores
Homo	1 845	N	1 145	27	12 112
		BFS	13	26	3 043
		DFS	27	27	6 937
		H	**12**	**25**	**2 364**
SacchCere	74 426	N	24 469	55	278 402
		BFS	**1 134**	**34**	89 883
		DFS	2 627	57	223 643
		H	1 146	35	**83 978**
DBLP	3 346	N	103 231	608	34 572
		BFS	68	612	6 184
		DFS	282	627	38 887
		H	**29**	**521**	**5 037**
Obama InIsrael	2 573	N	37 554	1 286	3 882
		BFS	226	1 299	3 313
		DFS	**150**	1 384	3 596
		H	177	**1 147**	**2 716**
Amazon	1 164	N	11 990	**425**	1 823
		BFS	3 981	534	1 354
		DFS	5 278	619	2 459
		H	**3 913**	536	**1 334**
Friendfeed Twitter	76 194	N	409 489	220	80 954
		BFS	61 113	**215**	80 664
		DFS	**1 973**	267	80 745
		H	59 520	268	**76 419**
Higgs	8 077	N	163 398	474	22 478
		BFS	2 480	**465**	12 773
		DFS	**640**	490	14 119
		H	2 169	493	**9 389**
Friendfeed	365 666	BFS	58 278	**465**	546 631
		DFS	**13 356**	591	568 107
		H	47 179	490	**389 323**

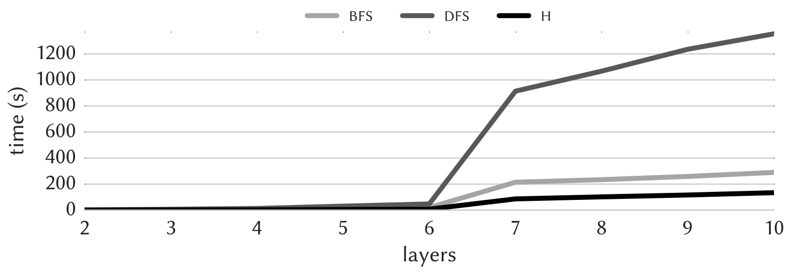


Figure 3.4: Runtime of the proposed methods with varying the number of layers (DBLP dataset).

Core-decomposition characterization. Figure 3.5 reports the distribution of number of cores, core size, and average-degree density (i.e., number of edges divided by number of vertices) of the subgraph corresponding to a core. Distributions are shown by level of the lattice[9] for the SacchCere and Friendfeed datasets. Although the two datasets have very different scales, the distributions exhibit similar trends. Being limited by the number of layers, the number of cores in the first levels

[9]Recall that the lattice level has been defined in Section 3.1: level i contains all cores whose coreness-vector components sum to i.

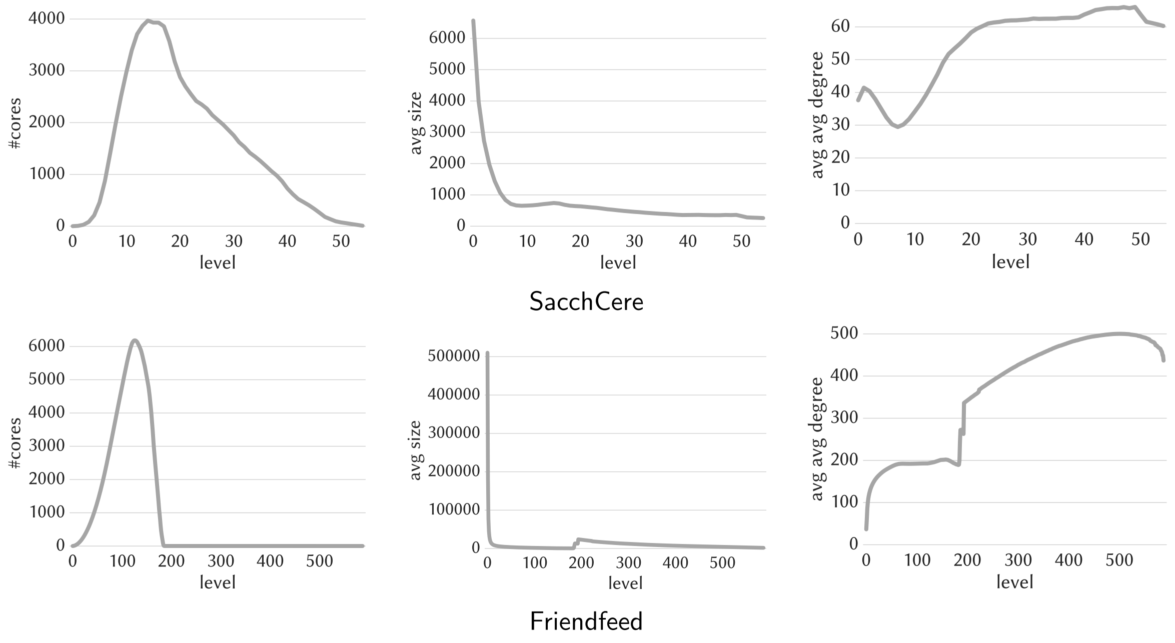


Figure 3.5: Distribution of number of cores (left), average core size (center), and average average-degree density of a core (right) to the core-lattice level, for datasets SacchCere (top) and Friendfeed (bottom).

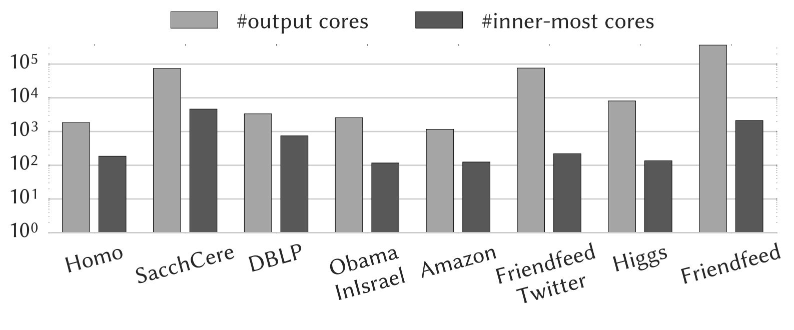


Figure 3.6: Number of output cores (total and inner-most).

of the lattice is very small, but then it exponentially grows until reaching its maximum within the first $25 - 30\%$ visited levels. The average size of the cores is close to the number of vertices in the first lattice level, when cores' degree conditions are not very strict. Then it decreases as the number of cores gets larger, with a maximum reached when very small cores stop "propagating" in the lower lattice levels. Finally, the average (average-degree) density tends to increase for higher lattice level. However, there are a couple of exceptions: it decreases (*i*) in the first few levels of SacchCere's lattice, and (*ii*) in the last levels of both SacchCere and Friendfeed, where the core size starts getting smaller, thus implying small average-degree values.

In Figure 3.6 we show the comparison between the number of all cores and inner-most cores for all the datasets. The number of cores differs quite a lot from dataset to dataset, depending on dataset size, number of layers, and density. The fraction of inner-most cores exhibits a non-decreasing trend as the layers increase, ranging from 0.3% of the total number of output cores (FriendfeedTwitter) to 22% (DBLP).

Given that the inner-most cores are *per-se* interesting and typically one or more orders of magnitude fewer in number than the total cores, it would be desirable to have a method that effectively exploits the maximality property and extracts the inner-most ones directly, without computing a complete decomposition. This is presented in the next section.

Algorithm 3.6: IM-ML-cores

```
  Input: A multilayer graph $G = (V, E, L)$.
  Output: The set $\mathbf{I}$ of all inner-most multilayer cores of $G$.
1 sort $L$ by non-decreasing average-degree density
2 $\mathcal{M} \leftarrow \emptyset$
3 $\mathbf{I} \leftarrow$ RIM-ML-cores$(G, V, [0]_{|L|}, \ell_1, \mathcal{M})$
```

3.3 Algorithms for inner-most multilayer cores

In this section we focus on the problem of finding the inner-most multilayer cores of a multilayer graph (Problem 3.2). Specifically, the main goal here is to devise a method that is more efficient than a naïve one that computes the whole multilayer core decomposition and then a-posteriori filters non-inner-most cores out. To this end, we devise a recursive algorithm, which is termed IM-ML-cores and whose outline is shown as Algorithm 3.6 (and Algorithm 3.7). We provide the details of the algorithm next. In the remainder of this section we assume the layer set L of the input multilayer graph $G = (V, E, L)$ to be an ordered list $[\ell_1, \ldots, \ell_{|L|}]$. The specific ordering we adopt in this work is by non-decreasing average-degree density, as, among the various orderings tested, this is the one that provides the best experimental results.

The proposed IM-ML-cores algorithm is based on the notion of *ℓ_r-right-inner-most multilayer cores* of a core $C_{\vec{k}}$, i.e., all those cores having coreness vector $\vec{k'}$ equal to $\vec{k}$ up to layer ℓ_{r-1}, and for which the inner-most condition holds for layers from ℓ_r to $\ell_{|L|}$.

Definition 3.4 (ℓ_r-right-inner-most multilayer cores)**.** *Given a multilayer graph $G = (V, E, L)$ and a layer $\ell_r \in L$, the* ℓ_r-right-inner-most multilayer cores *of a core $C_{\vec{k}}$ of G, where $\vec{k} = [k_\ell]_{\ell \in L}$, correspond to all the cores of G with coreness vector $\vec{k'} = [k'_\ell]_{\ell \in L}$ such that $\forall \ell \in [\ell_1, \ell_r) : k'_\ell = k_\ell$, and there does not exist any other core with coreness vector $\vec{k''} = [k''_\ell]_{\ell \in L}$ such that $\forall \ell \in [\ell_1, \ell_r) : k''_\ell = k_\ell$, $\forall \ell \in [\ell_r, \ell_{|L|}] : k''_\ell \geq k'_\ell$, and $\exists \hat{\ell} \in [\ell_r, \ell_{|L|}] : k''_{\hat{\ell}} > k'_{\hat{\ell}}$.*

Let $C_{[0]_{|L|}}$ be the root of the core lattice. $C_{[0]_{|L|}}$ has a coreness vector composed of zero components. Therefore, according to the above definition, it is easy to observe that the ℓ_1-right-inner-most multilayer cores of $C_{[0]_{|L|}}$ correspond to the desired ultimate output, i.e., to all inner-most multilayer cores of the input multilayer graph.

Fact 3.2. *Given a multilayer graph $G = (V, E, L)$, let $\mathbf{I}_{\ell_1}$ be the set of all ℓ_1-right-inner-most multilayer cores of core $C_{[0]_{|L|}}$. $\mathbf{I}_{\ell_1}$ corresponds to all inner-most multilayer cores of G.*

The proposed IM-ML-cores algorithm recursively computes ℓ_r-right-inner-most multilayer cores, starting from the root of the core lattice (Algorithm 3.6). The goal is to exploit Fact 3.2 and ultimately have the ℓ_1-right-inner-most multilayer cores of core $C_{[0]_{|L|}}$ computed. The algorithm makes use of a data structure $\mathcal{M}$ which consists of a sequence of nested maps, one for each layer but the last one (i.e., $\ell_{|L|}$). For every layer ℓ_r that has been so far processed by the recursive procedure, $\mathcal{M}$ keeps track of the minimum-degree that a core should have in layer ℓ_r to be recognized as an ineer-most one. Specifically, given a coreness vector $\vec{k}$ and a layer ℓ_r, the instruction $\mathcal{M}(\vec{k}, \ell_r)$ iteratively accesses the nested maps using the elements of $\vec{k}$ up to layer ℓ_r as keys. As an example, consider a coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$, with $|L| = 3$. $\mathcal{M}(\vec{k}, \ell_{|L|-1})$ first queries the outer-most map with key k_{ℓ_1}, and obtains a further map. Then, this second map is queried with key k_{ℓ_2}, to finally get the ultimate desired numerical value. Note that, if $\ell_r < \ell_{|L|-1}$, then $\mathcal{M}(\vec{k}, \ell_r)$ returns a further map. Conversely, if $\ell_r = \ell_{|L|-1}$, then $\mathcal{M}(\vec{k}, \ell_r)$ returns a numerical value. If $\vec{k}$ does not correspond to a sequence of valid keys for $\mathcal{M}$, we assume that 0 is returned as a default value. $\mathcal{M}$ is initialized as empty, and populated during the various recursive iterations.

Algorithm 3.7 may be logically split into two main blocks: the first one (Lines 3 – 7) taking care of the recursion, and the second one (Lines 9 – 19) computing the ℓ_r-right-inner-most cores. The first block of the algorithm is executed when the current ℓ_r layer is not the last one. In that block the $\vec{k}$-coresPath subroutine (already used in Algorithm 3.3 and described in Section 3.2.3) is run on set S of vertices, layer ℓ_r, and taking into account the constraints in vector $\vec{k}$ (Lines 3 and 4).

Algorithm 3.7: RIM-ML-cores

```
Input: A multilayer graph $G = (V, E, L)$, a set $S \subseteq V$ of vertices, a coreness vector
       $\vec{k} = [k_\ell]_{\ell \in L}$, a layer $\ell_r \in L$, and a data structure $\mathcal{M}$.
Output: The set $\mathbf{I}_r$ of all right-inner-most multilayer cores of $C_{\vec{k}}$ given $\ell_r$.
1  $\mathbf{I}_r \leftarrow \emptyset$
2  if $\ell_r \neq \ell_{|L|}$ then
3      $\mathbf{Q} \leftarrow \{\vec{k'} \mid C_{\vec{k'}} \in \vec{k}$-coresPath$(G, S, \vec{k}, \ell_r)\} \cup \{\vec{k}\}$
4      $\mathbf{C} \leftarrow \vec{k}$-coresPath$(G, S, \vec{k}, \ell_r) \cup \{S\}$
5      forall $\vec{k'} \in \mathbf{Q}$ in decreasing order of $k'_{\ell_r}$ do
6          $\mathcal{M}(\vec{k'}, \ell_r) \leftarrow \emptyset$
7          $\mathbf{I}_r \leftarrow \mathbf{I}_r \cup$ RIM-ML-cores$(G, C_{\vec{k'}}, \vec{k'}, \ell_{r+1}, \mathcal{M})$
8  else
9      $k_{\mathcal{M}} \leftarrow 0$
10     forall $\ell \in [\ell_1, \ell_{|L|})$ do
11         $\vec{k}^\ell = [k_{\ell_1}, \dots, k_\ell + 1, \dots, k_{\ell_{|L|}}]$
12         $k_{\mathcal{M}} \leftarrow \max\{k_{\mathcal{M}}, \mathcal{M}(\vec{k}^\ell, \ell_{|L|-1})\}$
13     $\vec{k'} \leftarrow [k_{\ell_1}, \dots, k_{\ell_{|L|-1}}, k_{\mathcal{M}}]$
14     $\vec{k}^I \leftarrow$ Inner-mostCore$(G, S, \vec{k'}, \ell_{|L|})$
15     if $\vec{k}^I \neq$ NULL then
16         $\mathbf{I}_r \leftarrow \mathbf{I}_r \cup \vec{k}^I$
17         $\mathcal{M}(\vec{k}^I, \ell_{|L|-1}) \leftarrow k^I_{\ell_{|L|}} + 1$
18     else
19         $\mathcal{M}(\vec{k'}, \ell_{|L|-1}) \leftarrow k'_{\ell_{|L|}}$
```

Then, for each coreness vector $\vec{k'}$ that has been found, a recursive call is made, where the layer of interest becomes the next layer ℓ_{r+1}, and the data structure $\mathcal{M}$ is augmented by adding a further (empty) nested map (this new map will be populated within the upcoming recursive executions).

The coreness vectors are processed in decreasing order of k'_{ℓ_r}. This processing order ensures the correctness of the following: once a multilayer core has been identified as ℓ_r-right-inner-most, it permanently becomes part of the ultimate output cores (no further recursive call will remove it from the output). Note also that, for each $\vec{k'}$, RIM-ML-cores can be run on $C_{\vec{k'}}$ only, i.e., the core of coreness vector $\vec{k'}$. This guarantees better efficiency, without affecting correctness.

The second block of the algorithm (Lines 9 – 19) works as follows. When the last layer has been reached, i.e., $\ell_r = \ell_{|L|}$, the current recursion ends, and an ℓ_r-right-inner-most multilayer core is returned (if any). First of all, the algorithm computes a coreness vector $\vec{k'}$ which is potentially ℓ_r-right-inner-most (Lines 9 – 13). In this regard, note that the $k_{\mathcal{M}}$ value is derived from the information that has been stored in $\mathcal{M}$ in the earlier recursive iterations. Finally, the algorithm computes the inner-most core in $\ell_{|L|}$ constrained by $\vec{k'}$, by means of the Inner-mostCore subroutine. Such a subroutine, similarly to the $\vec{k}$-coresPath one, takes as input a multilayer graph G, a subset S of vertices, a coreness vector $\vec{k}$, and a layer ℓ. It returns the multilayer core having coreness vector of highest ℓ-th component of the vertices in S, considering the constraints specified in $\vec{k}$. If the Inner-mostCore procedure actually returns a multilayer core, then it is guaranteed that such a core is ℓ_r-right-inner-most, and is therefore added to the solution (and $\mathcal{M}$ is updated accordingly).

In Figure 3.7 we show an example of the execution of the proposed IM-ML-cores algorithm for a simple 3-layer graph, while Figure 3.8 reports the content of the $\mathcal{M}$ data structure for this example. Every box corresponds to a call of Algorithm 3.7, for which we specify (i) the input parameters (G and $\mathcal{M}$ are omitted for the sake of brevity), (ii) the calls to the $\vec{k}$-coresPath or Inner-mostCore subroutines, and (iii) the content of $\mathbf{Q}$ (when it is instantiated). For instance, the coreness vector given as input to Inner-mostCore at box 1.3.4 has the last element equal to the maximum between

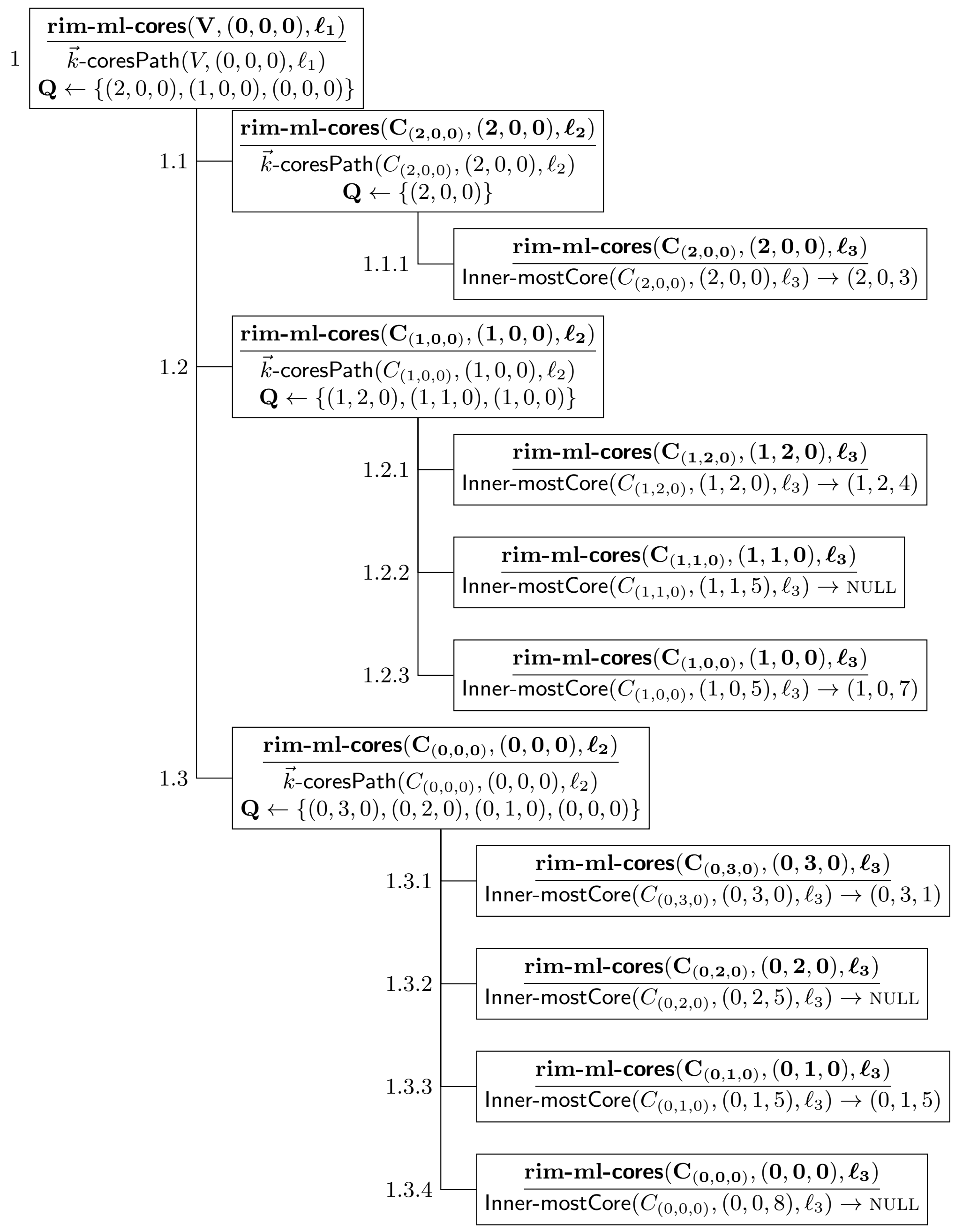

1 rim-ml-cores(V, (0,0,0), ℓ1)
k-coresPath(V, (0,0,0), ℓ1)
Q ← {(2,0,0), (1,0,0), (0,0,0)}
1.1 rim-ml-cores(C(2,0,0), (2,0,0), ℓ2)
k-coresPath(C(2,0,0), (2,0,0), ℓ2)
Q ← {(2,0,0)}
1.1.1 rim-ml-cores(C(2,0,0), (2,0,0), ℓ3)
Inner-mostCore(C(2,0,0), (2,0,0), ℓ3) → (2,0,3)
1.2 rim-ml-cores(C(1,0,0), (1,0,0), ℓ2)
k-coresPath(C(1,0,0), (1,0,0), ℓ2)
Q ← {(1,2,0), (1,1,0), (1,0,0)}
1.2.1 rim-ml-cores(C(1,2,0), (1,2,0), ℓ3)
Inner-mostCore(C(1,2,0), (1,2,0), ℓ3) → (1,2,4)
1.2.2 rim-ml-cores(C(1,1,0), (1,1,0), ℓ3)
Inner-mostCore(C(1,1,0), (1,1,5), ℓ3) → NULL
1.2.3 rim-ml-cores(C(1,0,0), (1,0,0), ℓ3)
Inner-mostCore(C(1,0,0), (1,0,5), ℓ3) → (1,0,7)
1.3 rim-ml-cores(C(0,0,0), (0,0,0), ℓ2)
k-coresPath(C(0,0,0), (0,0,0), ℓ2)
Q ← {(0,3,0), (0,2,0), (0,1,0), (0,0,0)}
1.3.1 rim-ml-cores(C(0,3,0), (0,3,0), ℓ3)
Inner-mostCore(C(0,3,0), (0,3,0), ℓ3) → (0,3,1)
1.3.2 rim-ml-cores(C(0,2,0), (0,2,0), ℓ3)
Inner-mostCore(C(0,2,0), (0,2,5), ℓ3) → NULL
1.3.3 rim-ml-cores(C(0,1,0), (0,1,0), ℓ3)
Inner-mostCore(C(0,1,0), (0,1,5), ℓ3) → (0,1,5)
1.3.4 rim-ml-cores(C(0,0,0), (0,0,0), ℓ3)
Inner-mostCore(C(0,0,0), (0,0,8), ℓ3) → NULL

Figure 3.7: Execution of the IM-ML-cores algorithm (Algorithm 3.6) on a toy 3-layer graph.

what is stored in $\mathcal{M}$ at the end of the paths $1 \to 0$ and $0 \to 1$, i.e., 8 and 5, that have been set at boxes 1.2.3 and 1.3.3, respectively.

3.3.1 Experimental results

Running time. We asses the efficiency of IM-ML-cores (for short IM) by comparing it to the aforementioned naïve approach for computing inner-most multilayer cores, which consists in firstly computing all multilayer cores (by means of one of the three algorithms presented in Section 3.2) and filtering out the non-inner-most ones. The results of this experiment are reported in Table 3.3. First of all, it can be observed that the a-posteriori filtering of the inner-most multilayer cores does not consistently affect the runtime of the algorithms for multilayer core decomposition: this means that most of the time is spent for computing the overall core decomposition. The main outcome of

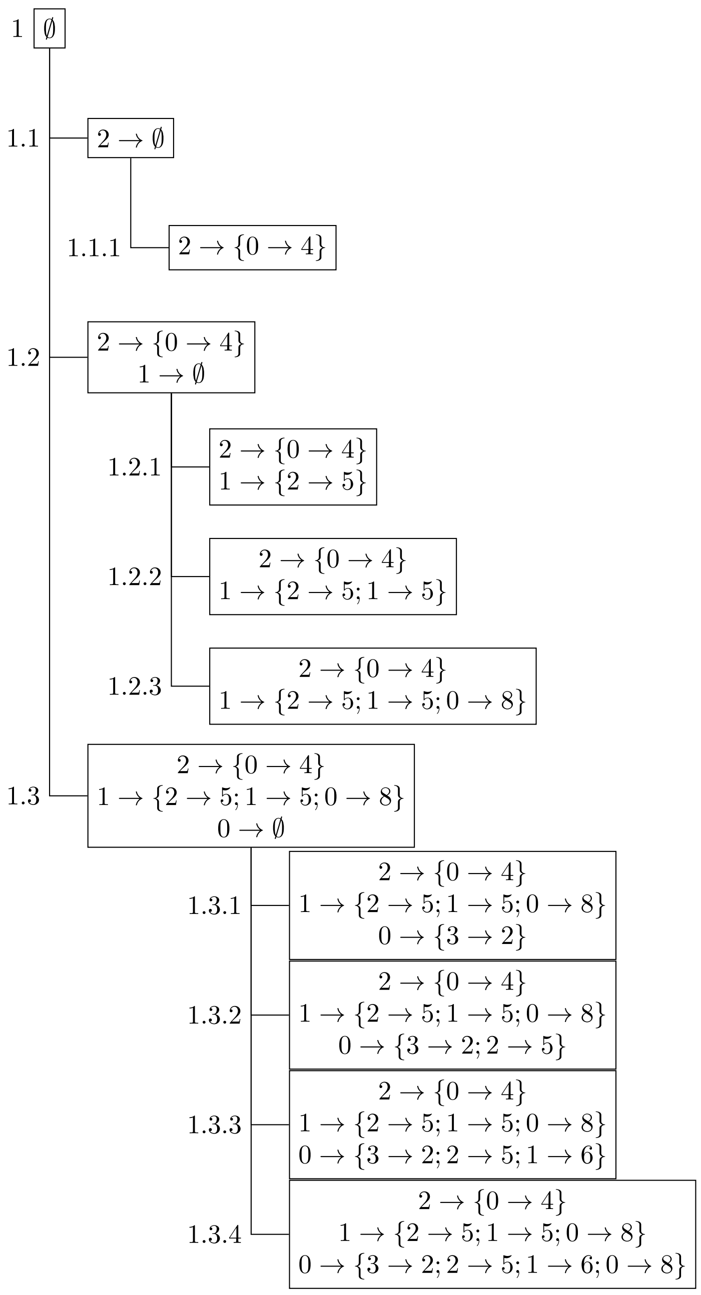


Figure 3.8: Content of the $\mathcal{M}$ data structure during the execution of the IM-ML-cores algorithm as per the example shown in Figure 3.7.

this experiment is that the running time of the proposed IM method is smaller than the time required by BFS, DFS, or H summed up to the time spent in the a-posteriori filtering, with considerable speed-up from 1.3 to an order of magnitude on the larger datasets, e.g., FriendfeedTwitter and Friendfeed. The only exception is on the DBLP dataset where BFS and H run slightly faster, probably due to fact that its edges are (almost) equally distributed among the layers, which makes the effectiveness of the ordering vanish.

Characterization. We also show the characteristics of the inner-most multilayer cores. Figure 3.9 reports the distribution of number, size, and average-degree density of all cores and inner-most cores only. Distributions are shown in a way similar to what previously done in Figure 3.5, i.e., by level of the core lattice, and for the SacchCere and Amazon datasets.

For both datasets, there are no inner-most cores in the first levels of the lattice. As expected,

Table 3.3: Runtime (in seconds) of the methods for multilayer core decomposition, the a-posteriori filtering of the inner-most multilayer cores, and the proposed IM-ML-cores method for directly computing inner-most multilayer cores.

dataset	BFS	DFS	H	filtering	IM
Homo	13	27	12	0.5	5
SacchCere	1 134	2 627	1 146	24	336
DBLP	68	282	29	1	148
ObamaInIsrael	226	150	177	7	120
Amazon	3 981	5 278	3 913	129	2 530
FriendfeedTwitter	61 113	1 973	59 520	276	1 583
Higgs	2 480	640	2 169	33	356
Friendfeed	58 278	13 356	47 179	394	2 640

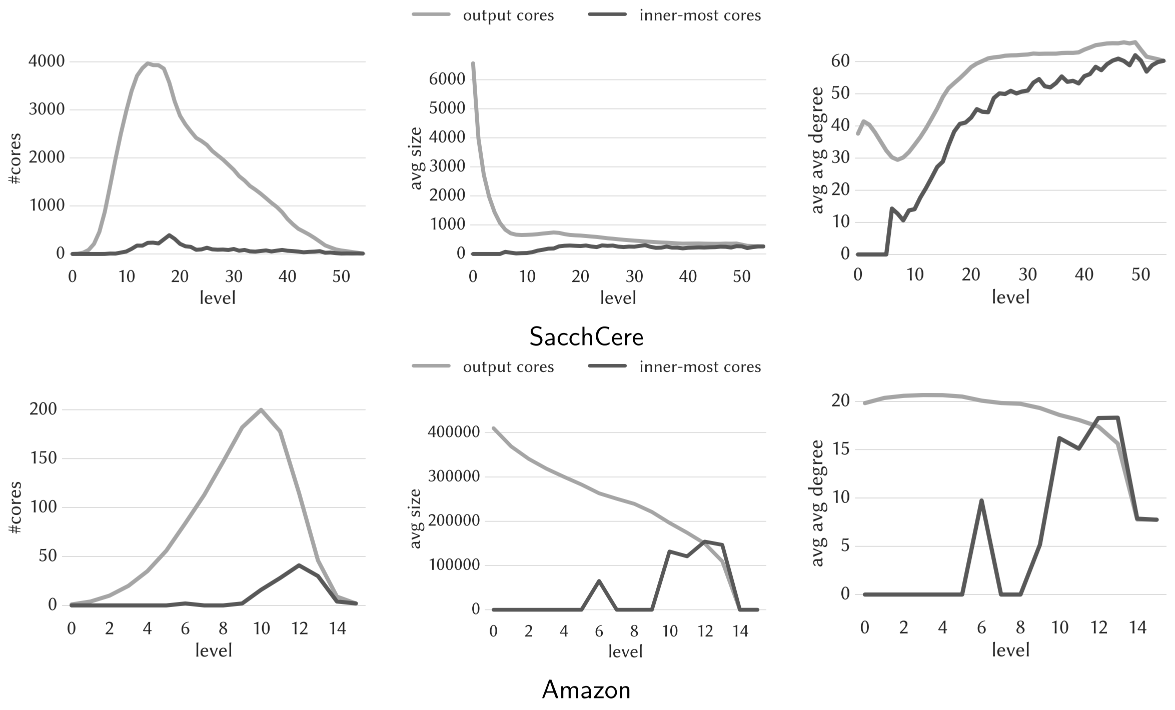


Figure 3.9: Comparison of the distributions, to the core-lattice level, of number (left), average size (center), and average average-degree density (right) of multilayer cores and inner-most multilayer cores, for datasets SacchCere (top) and Amazon (bottom).

the number of inner-most cores considerably increases when the number of all cores decreases. This is due to the fact that some cores stop propagating throughout the lattice, hence they are recognized as inner-most. In general, inner-most cores are on average smaller than all multilayer cores. Nonetheless, for the levels 12 and 13 of the Amazon dataset, inner-most cores have greater size than all cores. This behavior is consistent with our definitions: inner-most cores are cores without descendants, thus they are expected to be the smallest-sized ones, but they do not necessarily have to. Finally, the distribution of the average-degree density exhibits a similar trend to the distribution of the size: this is expected as the two measures depend on each other.

3.4 Multilayer densest subgraph

In this section we showcase the usefulness of multilayer core-decomposition in the context of multilayer densest-subgraph discovery. Particularly, we show how to exploit the multilayer core-decomposition to devise an algorithm with approximation guarantees for the

Multilayer Densest Subgraph problem introduced in Section 3.1 (Problem 3.3), thus extending to the multilayer setting the intuition at the basis of the well-known $\frac{1}{2}$-approximation algorithm [18, 55] for single-layer densest-subgraph extraction.

3.4.1 Hardness

We start by formally showing that the Multilayer Densest Subgraph problem (Problem 3.3) is **NP**-hard.

Theorem 3.3. *Problem 3.3 is* **NP***-hard.*

To prove the theorem, we introduce two variants of Problem 3.3's objective function, i.e., $\delta_{\text{ALL}}(\cdot)$, which considers all layers in L, and $\delta_{\neg\text{ALL}}(\cdot)$, which considers all subsets of layers but the whole layer set L. Specifically, for any given multilayer graph $G = (V, E, L)$ and vertex subset $S \subseteq V$, the two functions are defined as:

$$\delta_{\text{ALL}}(S) = \min_{\ell \in L} \frac{|E_\ell[S]|}{|S|} |L|^\beta, \tag{3.7}$$

$$\delta_{\neg\text{ALL}}(S) = \max_{\hat{L} \in 2^L \setminus \{L\}} \min_{\ell \in \hat{L}} \frac{|E_\ell[S]|}{|S|} |\hat{L}|^\beta. \tag{3.8}$$

We also define deg_{max} as the maximum degree of a vertex in a layer:

$$deg_{max} = \max_{\ell \in L} \max_{u \in V} deg(u, \ell), \tag{3.9}$$

and introduce the following three auxiliary lemmas.

Lemma 3.1. $\delta_{\text{ALL}}(S) \geq \frac{1}{|V|}|L|^\beta$*, for all* $S \subseteq V$ *such that* $\forall \ell \in L : |E_\ell[S]| > 0$.

Proof. For a vertex set S spanning at least one edge in every layer, it holds that $\min_{\ell \in L} \frac{|E_\ell[S]|}{|S|} \geq \frac{1}{|V|}$, and, therefore, $\delta_{\text{ALL}}(S) = \min_{\ell \in L} \frac{|E_\ell[S]|}{|S|}|L|^\beta \geq \frac{1}{|V|}|L|^\beta$. □

Lemma 3.2. $\delta_{\neg\text{ALL}}(S) \leq \frac{deg_{max}}{2}(|L| - 1)^\beta$*, for all* $S \subseteq V$.

Proof. The maximum density of a vertex set S in a layer can be at most equal to the density of the maximum clique, i.e., at most $\frac{(deg_{max}+1)\ deg_{max}}{2\ (deg_{max}+1)} = \frac{deg_{max}}{2}$. At the same time, the size of a layer set $\hat{L}$ in the function $\delta_{\neg\text{ALL}}(\cdot)$ can be at most $|L| - 1$ (as the whole layer set L is not considered in $\delta_{\neg\text{ALL}}(\cdot)$). This means that $\delta_{\neg\text{ALL}}(S) = \max_{\hat{L} \in 2^L \setminus \{L\}} \min_{\ell \in \hat{L}} \frac{|E_\ell[S]|}{|S|}|\hat{L}|^\beta \leq \frac{deg_{max}}{2}(|L| - 1)^\beta$. □

Lemma 3.3.

$$\beta > \frac{\log_{|L|-1}\left(\frac{|V|}{2} deg_{max}\right) \times \log_{|L|}(|L| - 1)}{1 - \log_{|L|}(|L| - 1)} \quad \Leftrightarrow \quad \frac{1}{|V|}|L|^\beta > \frac{deg_{max}}{2}(|L| - 1)^\beta. \tag{3.10}$$

Proof.

$$
\begin{aligned}
\beta &> \frac{\log_{|L|-1}\left(\frac{|V|}{2}deg_{max}\right) \times \log_{|L|}(|L|-1)}{1-\log_{|L|}(|L|-1)} && (3.11)\\
\Leftrightarrow\quad \left(1-\log_{|L|}(|L|-1)\right)\beta &> \log_{|L|-1}\left(\frac{|V|}{2}deg_{max}\right) \times \log_{|L|}(|L|-1) && (3.12)\\
\Leftrightarrow\quad \beta &> \log_{|L|-1}\left(\frac{|V|}{2}deg_{max}\right) \times \log_{|L|}(|L|-1) + \beta\log_{|L|}(|L|-1) && (3.13)\\
\Leftrightarrow\quad \frac{\beta}{\log_{|L|}(|L|-1)} &> \log_{|L|-1}\left(\frac{|V|}{2}deg_{max}\right) + \beta && (3.14)\\
\Leftrightarrow\quad \frac{\log_{|L|}|L|^{\beta}}{\log_{|L|}(|L|-1)} &> \log_{|L|-1}\left(\frac{|V|}{2}deg_{max}\right) + \log_{|L|-1}(|L|-1)^{\beta} && (3.15)\\
\Leftrightarrow\quad \log_{|L|-1}|L|^{\beta} &> \log_{|L|-1}\left(\frac{|V|}{2}deg_{max}(|L|-1)^{\beta}\right) && (3.16)\\
\Leftrightarrow\quad |L|^{\beta} &> \frac{|V|}{2}deg_{max}(|L|-1)^{\beta} && (3.17)\\
\Leftrightarrow\quad \frac{1}{|V|}|L|^{\beta} &> \frac{deg_{max}}{2}(|L|-1)^{\beta}. && (3.18)
\end{aligned}
$$

□

With Lemmas 3.1–3.3 in place, we are now ready to provide the ultimate proof of Theorem 3.3.

Proof. We reduce from the Min-Avg Densest Common Subgraph (DCS-MA) problem [136], which aims at finding a subset of vertices $S \subseteq V$ from a multilayer graph $G = (V, L, S)$ maximizing $\min_{\ell \in L} \frac{E_\ell[S]}{|S|}$, and has been recently shown to be **NP**-hard in [56]. We distinguish two cases. The first (trivial) one is when G has a layer with no edges. In this case any vertex subset would be an optimal solution for DCS-MA (with overall objective function equal to zero), including the optimal solution to our Multilayer Densest Subgraph problem run on the same G (no matter which β is used). In the second case G has at least one edge in every layer. In this case solving our Multilayer Densest Subgraph problem on G, with β set to any value $> \frac{\log_{|L|-1}\left(\frac{|V|}{2}deg_{max}\right)\times\log_{|L|}(|L|-1)}{1-\log_{|L|}(|L|-1)}$, gives a solution that is optimal for DCS-MA as well. Indeed, it can be observed that, for all $S \subseteq V$ such that $\forall \ell \in L : |E_\ell[S]| > 0$:

$$
\begin{aligned}
\delta_{\text{ALL}}(S) &\geq \frac{1}{|V|}|L|^{\beta} && \{\text{Lemma 3.1}\} && (3.19)\\
&> \frac{deg_{max}}{2}(|L|-1)^{\beta} && \{\text{Lemma 3.3}\} && (3.20)\\
&\geq \delta_{\neg\text{ALL}}(S). && \{\text{Lemma 3.2}\} && (3.21)
\end{aligned}
$$

This means that, for that particular value of β, the optimal solution of Multilayer Densest Subgraph on input G is given by maximizing the $\delta_{\text{ALL}}(\cdot)$ function, which considers all layers and is, as such, equivalent to the objective function underlying the DCS-MA problem. This completes the proof. □

3.4.2 Algorithms

The approximation algorithm we devise for the Multilayer Densest Subgraph problem is very simple: it computes the multilayer core decomposition of the input graph, and, among all cores, takes the one maximizing the objective function δ as the output densest subgraph (Algorithm 3.8). Despite its simplicity, the algorithm achieves provable approximation guarantees proportional to the number of layers of the input graph, precisely equal to $\frac{1}{2|L|^{\beta}}$. We next formally prove this result.

Let $\mathbf{C}$ be the core decomposition of the input multilayer graph $G = (V, E, L)$ and C^* denote the core in $\mathbf{C}$ maximizing the density function δ, i.e., $C^* = \arg\max_{C \in \mathbf{C}} \delta(C)$. Then, C^* corresponds to the subgraph output by the proposed ml-densest algorithm. Let also $C^{(\mu)}$ denote the subgraph

Algorithm 3.8: ML-densest

```
Input: A multilayer graph $G = (V, E, L)$ and a real number $\beta \in \mathbb{R}^+$.
Output: $C^* \subseteq V$.
1 $\mathbf{C} \leftarrow$ MultiLayerCoreDecomposition($G$)    // Any of Algorithms 3.2, 3.3, 3.5 can be used
2 $C^* \leftarrow \arg\max_{C \in \mathbf{C}} \delta(C)$    // Equation (3.2)
```

maximizing the minimum degree in a single layer, i.e., $C^{(\mu)} = \arg\max_{S \subseteq V} f(S)$, where $f(S) = \max_{\ell \in L} \mu(S, \ell)$, while $\ell^{(\mu)} = \arg\max_{\ell \in L} \mu(C^{(\mu)}, \ell)$. It is easy to see that $C^{(\mu)} \in \mathbf{C}$. Finally, let S^*_{SL} be the densest subgraph among all single-layer densest subgraphs, i.e., $S^*_{\mathrm{SL}} = \arg\max_{S \subseteq V} g(S)$, where $g(S) = \max_{\ell \in L} \frac{|E_\ell[S]|}{|S|}$, and ℓ^* be the layer where S^*_{SL} exhibits its largest density, i.e., $\ell^* = \arg\max_{\ell \in L} \frac{|E_\ell[S^*_{\mathrm{SL}}]|}{|S^*_{\mathrm{SL}}|}$. We start by introducing the following two lemmas that can straightforwardly be derived from the definitions of C^*, $C^{(\mu)}$, S^*_{SL}, $\ell^{(\mu)}$, and ℓ^*:

Lemma 3.4. $\delta(C^*) \geq \delta(C^{(\mu)})$.

Proof. By definition, $C^{(\mu)}$ is a multilayer core described by (among others) the coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$ with $k_{\ell^{(\mu)}} = \max_{\ell \in L} \mu(C^{(\mu)}, \ell)$, and $k_\ell = 0$, $\forall \ell \neq \ell^{(\mu)}$. Then $C^{(\mu)} \in \mathcal{C}$. As $C^* = \arg\max_{C \in \mathcal{C}} \delta(C)$, it holds that $\delta(C^*) \geq \delta(C^{(\mu)})$. □

Lemma 3.5. $\delta(S^*) \leq \frac{|E_{\ell^*}[S^*_{\mathrm{SL}}]|}{|S^*_{\mathrm{SL}}|}|L|^\beta$.

Proof.

$$\delta(S^*) \;=\; \max_{\hat{L} \subseteq L} \min_{\ell \in \hat{L}} \frac{|E_\ell[S^*]|}{|S^*|}|\hat{L}|^\beta \;\leq\; \max_{\ell \in L} \frac{|E_\ell[S^*]|}{|S^*|}|L|^\beta \;\leq\; \frac{|E_{\ell^*}[S^*_{\mathrm{SL}}]|}{|S^*_{\mathrm{SL}}|}|L|^\beta. \tag{3.22}$$

□

The following further lemma shows a lower bound on the minimum degree of a vertex in S^*_{SL}:

Lemma 3.6. $\mu(S^*_{\mathrm{SL}}, \ell^*) \geq \frac{|E_{\ell^*}[S^*_{\mathrm{SL}}]|}{|S^*_{\mathrm{SL}}|}$.

Proof. As S^*_{SL} is the subgraph maximizing the density in layer ℓ^*, removing the minimum-degree node from S^*_{SL} cannot increase that density. Thus, it holds that:

$$\frac{|E_{\ell^*}[S^*_{\mathrm{SL}}]|}{|S^*|} \geq \frac{|E_{\ell^*}[S^*_{\mathrm{SL}}]| - \mu(S^*_{\mathrm{SL}}, \ell^*)}{|S^*_{\mathrm{SL}}| - 1} \tag{3.23}$$

$$\Leftrightarrow \quad \mu(S^*_{\mathrm{SL}}, \ell^*) \geq |E_{\ell^*}[S^*_{\mathrm{SL}}]| \frac{|S^*_{\mathrm{SL}}| - 1}{|S^*_{\mathrm{SL}}|} - |E_{\ell^*}[S^*_{\mathrm{SL}}]| \tag{3.24}$$

$$\Leftrightarrow \quad \mu(S^*_{\mathrm{SL}}, \ell^*) \geq \frac{|E_{\ell^*}[S^*_{\mathrm{SL}}]|}{|S^*_{\mathrm{SL}}|}. \tag{3.25}$$

□

The approximation factor of the proposed ML-densest algorithm is ultimately stated in the next theorem:

Theorem 3.4. $\delta(C^*) \geq \frac{1}{2|L|^\beta}\delta(S^*)$.

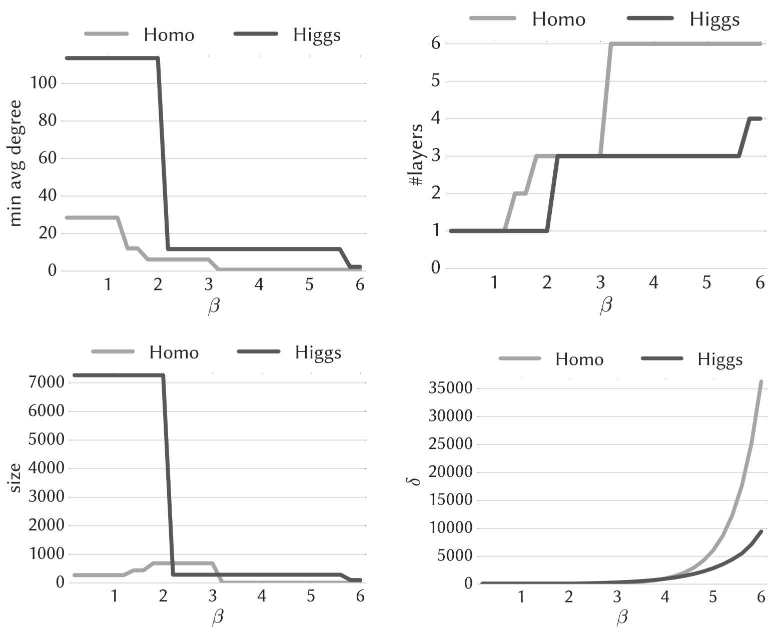


Figure 3.10: Multilayer densest-subgraph extraction (Homo and Higgs datasets): minimum average-degree density in a layer, number of selected layers, size, and objective-function value δ of the output densest subgraphs with varying β.

Proof.

$$
\begin{aligned}
\delta(C^*) &\geq \delta(C^{(\mu)}) && \{\text{Lemma 3.4}\} && (3.26)\\
&\geq \max_{\ell \in L} \frac{|E_\ell[C^{(\mu)}]|}{|C^{(\mu)}|} 1^\beta = \max_{\ell \in L} \frac{|E_\ell[C^{(\mu)}]|}{|C^{(\mu)}|} && \{\text{Equation (3.2)}\} && (3.27)\\
&\geq \frac{1}{2} \max_{\ell \in L} \mu(C^{(\mu)}, \ell) && \{\text{as avg degree} \geq \text{min degree}\} && (3.28)\\
&= \frac{1}{2} \mu(C^{(\mu)}, \ell^{(\mu)}) && \{\text{by definition of } C^{(\mu)}\} && (3.29)\\
&\geq \frac{1}{2} \mu(S^*_{\mathrm{SL}}, \ell^*) && \{\text{optimality of } C^{(\mu)} \text{ w.r.t. min degree}\} && (3.30)\\
&\geq \frac{1}{2} \frac{|E_{\ell^*}[S^*_{\mathrm{SL}}]|}{|S^*_{\mathrm{SL}}|} && \{\text{Lemma 3.6}\} && (3.31)\\
&\geq \frac{1}{2|L|^\beta} \delta(S^*). && \{\text{Lemma 3.5}\} && (3.32)
\end{aligned}
$$

□

The following corollary shows that the theoretical approximation guarantee stated in Theorem 3.4 remains the same even if only the inner-most cores are considered (although, clearly, considering the whole core decomposition may lead to better accuracy in practice).

Corollary 3.4. *Given a multilayer graph $G = (V, E, L)$, let $\mathbf{C}_{\mathrm{IM}}$ be the set of all inner-most multilayer cores of G, and let $C^*_{\mathrm{IM}} = \arg\max_{C \in \mathbf{C}_{\mathrm{IM}}} \delta(C)$. It holds that $\delta(C^*_{\mathrm{IM}}) \geq \frac{1}{2|L|^\beta} \delta(S^*)$.*

Proof. Let $C^{(\mu)}_{\mathrm{IM}} \in \mathbf{C}_{\mathrm{IM}}$ be an inner-most core of G whose coreness vector has a component equal to $\ell^{(\mu)}$. It is easy to see that the result in Lemma 3.4 holds for C^*_{IM} and $C^{(\mu)}_{\mathrm{IM}}$ too, i.e., becoming $\delta(C^*_{\mathrm{IM}}) \geq \delta(C^{(\mu)}_{\mathrm{IM}})$, while the proof of Theorem 3.4 holds as is, by simply replacing C^* with C^*_{IM} and $C^{(\mu)}$ with $C^{(\mu)}_{\mathrm{IM}}$. □

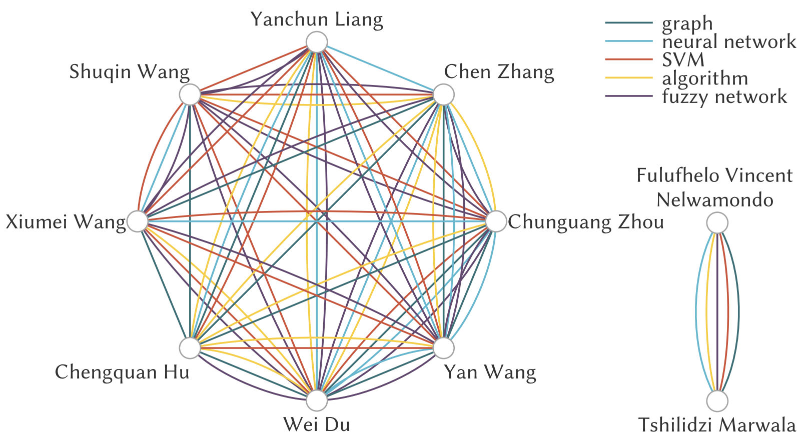


Figure 3.11: Multilayer densest subgraph extracted by Algorithm 3.8 from the DBLP dataset ($\beta = 2.2$).

Finally, we observe that the result in Theorem 3.4 carries over to the Min-Avg Densest Common Subgraph (DCS-MA) problem studied in [56, 136, 195, 210] as well, as that problem can be reduced to our Multilayer Densest Subgraph problem (as shown in Theorem 3.3).

3.4.3 Experimental results

We experimentally evaluate our ML-densest algorithm (Algorithm 3.8) on the datasets in Table 3.1. Figure 3.10 reports the results – minimum average-degree density in a layer, number of selected layers, size, objective-function value δ – on the Homo and Higgs datasets, with varying β. The remaining datasets, which we omit due to space constraints, exhibit similar trends on all measures.

The trends observed in the figure conform to what expected: the smaller β, the more the objective function privileges solutions with large average-degree density in a few layers (or even just one layer, for β close to zero). The situation is overturned with larger values of β, where the minimum average-degree density drops significantly, while the number of selected layers stands at 6 for Homo and 4 for Higgs. In-between β values lead to a balancing of the two terms of the objective function, thus giving more interesting solutions. Also, by definition, δ as a function of β draws exponential curves.

Finally, as anecdotal evidence of the output of Algorithm 3.8, in Figure 3.11 we report the densest subgraph extracted from DBLP. The subgraph contains 10 vertices and 5 layers automatically selected by the objective function δ. The minimum average-degree density is encountered on the layers corresponding to topics "*graph*" and "*algorithm*" (green and yellow layers in the figure), and is equal to 1.2. The objective-function value is $\delta = 41.39$. Note that the subgraph is composed of two connected components. In fact, like the single-layer case, multilayer cores are not necessarily connected.

3.5 Multilayer quasi-cliques

Another interesting insight into the notion of multilayer cores is about their relationship with (quasi-)cliques. In single-layer graphs it is well-known that cores can be exploited to speed-up the problem of finding cliques, as a clique of size k is guaranteed to be contained into the $(k-1)$-core. Interestingly, a similar relationship holds in the multilayer context too. Given a multilayer graph $G = (V, E, L)$, a layer $\ell \in L$, and a real number $\gamma \in (0, 1]$, a subgraph $G[S] = (S \subseteq V, E[S], L)$ of G is said to be a γ-*quasi-clique* in layer ℓ if all its vertices have at least $\gamma(|S| - 1)$ neighbors in layer ℓ within S, i.e., $\forall u \in S : deg_S(u, \ell) \geq \gamma(|S| - 1)$. Jiang et al. [137] study the problem of extracting *frequent cross-graph quasi-cliques*:[10] given a multilayer graph $G = (V, E, L)$, a function

[10]The input in [137] has the form of a set of graphs sharing the same vertex set, which is clearly fully equivalent to the notion of multilayer graph considered in this work.

Table 3.4: Comparison of the runtime of the efficient extraction of frequent cross-graph quasi-cliques by Corollary 3.5 and of the original algorithm [137], for the SacchCere dataset. The evaluation is proposed varying one of the parameters, i.e., Γ, *min_sup*, and *min_size*, at a time. The number of solution quasi-cliques and the number of vertices $|V'|$ of the subgraph G' are also reported.

Γ	*min_sup*	*min_size*	# solution quasi-cliques	$\|V'\|$	runtime (s) Corollary 3.5	runtime (s) [137]
1 1 1 1 .2 .2 1	0.5	6	2	371	3	169
.9 .9 .9 .9 .2 .2 .9			2	371	25	17 561
.8 .8 .8 .8 .2 .2 .8			6	1 196	734	22 932
.7 .7 .7 .7 .2 .2 .7			6	1 196	728	23 376
.6 .6 .6 .6 .2 .2 .6			59	2 300	5 200	28 948
.5 .5 .5 .5 .2 .2 .5			59	2 300	5 123	29 677

Γ	*min_sup*	*min_size*	# solution quasi-cliques	$\|V'\|$	runtime (s) Corollary 3.5	runtime (s) [137]
.5 .5 .5 .5 .2 .2 .5	1	3	2	152	2	281
	0.9		2	152	2	282
	0.8		28	940	23	292
	0.7		323	3 271	205	411
	0.6		323	3 271	203	414
	0.5		1 630	4 581	2 569	3 075

Γ	*min_sup*	*min_size*	# solution quasi-cliques	$\|V'\|$	runtime (s) Corollary 3.5	runtime (s) [137]
.5 .5 .5 .5 .2 .2 .5	0.5	7	27	2 254	5 606	34 904
		6	59	2 300	5 123	29 677
		5	357	3 363	4 493	21 206
		4	378	3 363	3 704	15 465
		3	1 630	4 581	2 569	3 075

$\Gamma : L \to (0, 1]$ assigning a real value to every layer in L, a real number *min_sup* $\in (0, 1]$, and an integer *min_size* > 1, find all maximal subgraphs $G[S]$ of G of size larger than *min_size* such that there exist at least *min_sup* $\times |L|$ layers ℓ for which $G[S]$ is a $\Gamma(\ell)$-quasi-clique.

The following theorem shows that a frequent cross-graph quasi-clique of size $\geq$ *min_size* is necessarily contained into a $\vec{k}$-core described by a coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$ such that there exists a fraction of *min_sup* layers ℓ where $k_\ell = \lceil \Gamma(\ell)(\mathit{min\_size} - 1) \rceil$.

Theorem 3.5. *Given a multilayer graph $G = (V, E, L)$, a real-valued function $\Gamma : L \to (0, 1]$, a real number* min_sup $\in (0, 1]$*, and an integer* min_size > 1*, a frequent cross-graph quasi-clique of G complying with parameters Γ,* min_sup*, and* min_size *is contained into a $\vec{k}$-core with coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$ such that $|\{\ell \in L : k_\ell = \lceil \Gamma(\ell)(\text{min\_size} - 1) \rceil\}| = \lceil \text{min\_sup} \times |L| \rceil$.*

Proof. Assume that a cross-graph quasi-clique S of G complying with parameters Γ, *min_sup*, and *min_size* is not contained into any $\vec{k}$-core with coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$ such that $|\{\ell \in L : k_\ell = \lceil \Gamma(\ell)(\mathit{min\_size} - 1) \rceil\}| = \lceil \mathit{min\_sup} \times |L| \rceil$. This means that S contains a vertex u such that $|\{\ell \in L : deg_S(u, \ell) \geq \Gamma(\ell)(\mathit{min\_size} - 1)\}| < \mathit{min\_sup} \times |L|$, which means that $|\{\ell \in L : deg_S(u, \ell) \geq \Gamma(\ell)(|S| - 1)\}| < \mathit{min\_sup} \times |L|$ as well, since $|S| \geq \mathit{min\_size}$. This violates the definition of frequent cross-graph quasi-clique. □

As a simple corollary, the computation of frequent cross-graph quasi-cliques can therefore be circumstantiated to the subgraph given by the union of all multilayer cores complying with the condition stated in Theorem 3.5.

Corollary 3.5. *Given a multilayer graph $G = (V, E, L)$, a real-valued function $\Gamma : L \to (0, 1]$, a real number* min_sup $\in (0, 1]$*, and an integer* min_size > 1*, let $G' = (V', E', L)$ the subgraph of*

Table 3.5: Comparison of the runtime of the efficient extraction of frequent cross-graph quasi-cliques by Corollary 3.5 and of the original algorithm [137], for the DBLP dataset. The evaluation is proposed varying one of the parameters, i.e., Γ, *min_sup*, and *min_size*, at a time. The number of solution quasi-cliques and the number of vertices $|V'|$ of the subgraph G' are also reported. ++ indicates runtime longer than 259 200 seconds (i.e., 3 days).

Γ	*min_sup*	*min_size*	# solution quasi-cliques	$\|V'\|$	runtime (s) Corollary 3.5	runtime (s) [137]
1 1 1 1 1 1 1 1 1 1	0.2	8	2	18	0.2	26 496
.9 .9 .9 .9 .9 .9 .9 .9 .9 .9			2	18	0.2	26 112
.8 .8 .8 .8 .8 .8 .8 .8 .8 .8			13	75	0.3	26 867
.7 .7 .7 .7 .7 .7 .7 .7 .7 .7			18	196	1	27 387
.6 .6 .6 .6 .6 .6 .6 .6 .6 .6			18	196	1	27 084
.5 .5 .5 .5 .5 .5 .5 .5 .5 .5			121	801	18	31 508

Γ	*min_sup*	*min_size*	# solution quasi-cliques	$\|V'\|$	runtime (s) Corollary 3.5	runtime (s) [137]
.5 .5 .5 .5 .5 .5 .5 .5 .5 .5	0.5	3	8	182	0.2	26 969
	0.4		195	2 375	1	26 964
	0.3		3 394	22 659	210	32 981

Γ	*min_sup*	*min_size*	# solution quasi-cliques	$\|V'\|$	runtime (s) Corollary 3.5	runtime (s) [137]
.5 .5 .5 .5 .5 .5 .5 .5 .5 .5	0.2	13	1	75	0.2	26 644
		12	1	75	0.2	27 136
		11	8	196	0.7	26 966
		10	10	196	0.7	27 116
		9	116	801	18	32 372
		8	121	801	18	31 508
		7	1 292	3 468	181	113 558
		6	1 370	3 468	198	113 520
		5	7 599	15 316	3 790	++
		4	8 578	15 316	3 502	++

G given by the union of all multilayer cores of G complying with Theorem 3.5. It holds that all cross-graph quasi-cliques of G complying with parameters Γ, min_sup, *and* min_size *are contained into G'.*

The finding in Corollary 3.5 can profitably be exploited to have a more efficient extraction of frequent cross-graph quasi-cliques. Specifically, the idea is to (*i*) compute *all* multilayer cores of the input graph G (including the non-distinct ones, as the condition stated in Theorem 3.5 refers to not necessarily maximal coreness vectors); (*ii*) process all multilayer cores of G one by one, retain only the ones complying with Theorem 3.5, and compute the subgraph G' induced by the union of all such cores; (*iii*) run any algorithm for frequent cross-graph quasi-cliques on G'. Based on the above theoretical results, such a procedure is guaranteed to be sound and complete, and it is expected to provide a significant speed-up, as G' is expected to be much smaller than the original graph G.

3.5.1 Experimental results

We show in Tables 3.4 and 3.5 the experimental results about the comparison of the algorithm proposed by Jiang et al. [137] and the more efficient extraction of frequent cross-graph quasi-cliques by Corollary 3.5. Table 3.4 refers to the SacchCere dataset, while Table 3.5 to the DBLP dataset. To evaluate the effect of the parameters, i.e., the function Γ, *min_sup*, and *min_size*, on the performance of the two approaches, we vary a parameter at a time keeping the other two fixed. With regards to the values selected for Γ, we fix $\Gamma(\ell_5) = \Gamma(\ell_6) = 0.2$ in all the experiments involving the SacchCere dataset, due to the imbalance of the distribution of the edges in favor of the other five layers (i.e., layers $\ell_1, \ldots, \ell_4, \ell_7$). Instead, given the uniformity of the edge density across the layers of the DBLP dataset, Γ is modified coherently for all the layer in this latter case. In addition to the execution times, for each configuration of the parameters, we also report the number of solution frequent cross-graph quasi-cliques and the number of vertices $|V'|$ of the subgraph G' identified by Corollary 3.5.

The first thing to notice is that, in both datasets and for every configuration, our approach is faster than the algorithm by Jiang et al. [137]. The actual speed-up varies with the size of $|V'|$ (with respect to $|V|$) which, in turn, is affected by the mining parameters. For the SacchCere dataset, we obtain the most extreme cases when varying *min_sup* (middle table): our approach is able to prune from 30% (*min_sup* $= 0.5$) up to 98% (*min_sup* $= 1$) of the input multilayer graph. For the DBLP dataset, the results are even stronger: in the worst case (i.e., $\Gamma(\ell) = 0.5\ \forall \ell \in L$, *min_sup* $= 0.3$, and *min_size* $= 3$) we prune the 95% of the original vertex set. The runtime of both our approach and Jiang et al.'s [137] algorithm varies consistently according to parameters and to $|V'|$. The speed-up that our method reaches ranges from 1.2 to two orders of magnitude for the SacchCere dataset, and from one order up to six orders of magnitude for the DBLP dataset.

3.6 Multilayer community search

The idea here is very similar to that of the multilayer densest subgraph.

Problem 3.5 (Multilayer Community Search). *Given a multilayer graph $G = (V, E, L)$, a set of vertices $S \subseteq V$, and a set of layers $\hat{L} \subseteq L$, we define the minimum degree of a vertex in S, within the subgraph induced by S and $\hat{L}$ as:*

$$\varphi(S, \hat{L}) = \min_{\ell \in \hat{L}} \min_{u \in S} deg_S(u, \ell). \tag{3.33}$$

Given a positive real number β, we define a real-valued density function $\vartheta : 2^V \to \mathbb{R}^+$ as:

$$\vartheta(S) = \max_{\hat{L} \subseteq L} \varphi(S, \hat{L}) |\hat{L}|^\beta. \tag{3.34}$$

Given a set $V_Q \subseteq V$ of query vertices, find a subgraph containing all the query vertices and maximizing the density function, i.e.,

$$S^* = \arg\max_{V_Q \subseteq S \subseteq V} \vartheta(S). \tag{3.35}$$

Let $\mathbf{C}$ be the set of all non-empty multilayer cores of G. For a core $C \in \mathbf{C}$ with coreness vector $\vec{k} = [k_\ell]_{\ell \in L}$, we define the score

$$\sigma(C) = \max_{\hat{L} \subseteq L} (\min_{\ell \in \hat{L}} k_\ell) |\hat{L}|^\beta, \tag{3.36}$$

and denote by C^* a core that contains all query vertices in V_Q and maximizes the score σ, i.e.,

$$C^* = \arg\max_{C \in \mathbf{C}, V_Q \subseteq C} \sigma(C). \tag{3.37}$$

As shown in the following theorem, C^* is a (not necessarily unique) *exact* solution to Problem 3.5.

Theorem 3.6. *Given a multilayer graph $G = (V, E, L)$, and a set $V_Q \subseteq V$ of query vertices, let S^* and C^* be the vertex sets defined as in Equation (3.35) and Equation (3.37), respectively. It holds that $\vartheta(C^*) = \vartheta(S^*)$.*

Proof. We prove the statement by contradiction, assuming that $\vartheta(C^*) < \vartheta(S^*)$. Let $\mu_\ell = \min_{u \in S^*} deg_{S^*}(u, \ell)$, and $\vec{\mu} = [\mu_\ell]_{\ell \in L}$. By definition of multilayer core, there exists a core $C \in \mathbf{C}$ of G with coreness vector $\vec{\mu}$ such that $S^* \subseteq C$. This means that

$$\sigma(C) = \max_{\hat{L} \subseteq L} (\min_{\ell \in \hat{L}} \mu_\ell) |\hat{L}|^\beta = \max_{\hat{L} \subseteq L} (\min_{\ell \in \hat{L}} \min_{u \in S^*} deg_{S^*}(u, \ell)) |\hat{L}|^\beta = \vartheta(S^*). \tag{3.38}$$

Thus, there exists a core $C \in \mathbf{C}$ whose $\vartheta(\cdot)$ score is equal to $\vartheta(S^*)$, which contradicts the original assumption $\vartheta(C^*) < \vartheta(S^*)$. □

Table 3.6: Comparison of the average runtime (in seconds) between the original algorithms for multilayer core decomposition and modified methods for community search, with varying the number $|V_Q|$ of query vertices. In each dataset and for each $|V_Q|$, the smallest runtime is bolded.

			$\|V_Q\|$									
dataset	method	original	1	2	3	4	5	6	7	8	9	10
Homo	BFS	13	2	1	0.7	0.7	0.6	0.6	0.6	0.6	0.6	0.6
	DFS	27	3	2	1	1	1	0.9	0.9	0.9	0.9	0.9
	H	12	**0.9**	**0.3**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
SacchCere	BFS	1 134	**162**	**25**	6	3	1	1	0.7	0.7	0.5	0.5
	DFS	2 627	390	58	13	6	2	2	1	1	0.7	0.6
	H	1 146	166	**25**	**5**	**2**	**0.5**	**0.8**	**0.2**	**0.2**	**0.1**	**0.1**
DBLP	BFS	68	35	35	34	34	34	34	35	34	35	36
	DFS	282	55	42	39	39	38	38	38	38	39	39
	H	29	**5**	**5**	**5**	**5**	**5**	**6**	**6**	**6**	**6**	**6**
Obama	BFS	226	42	36	34	33	31	32	32	32	32	33
InIsrael	DFS	150	51	38	34	33	31	31	31	30	31	31
	H	177	**15**	**10**	**10**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
Amazon	BFS	3 981	2 125	1 364	608	582	441	234	231	192	175	167
	DFS	5 278	3 103	2 105	1 198	1 072	851	523	515	434	406	371
	H	3 913	**2 109**	**1 342**	**570**	**546**	**405**	**190**	**190**	**150**	**134**	**127**
Friendfeed	BFS	61 113	2 464	1 004	597	333	243	185	117	108	85	59
Twitter	DFS	1 973	**129**	**73**	**48**	**33**	**30**	**27**	**22**	**21**	**19**	**17**
	H	59 520	2 340	916	523	278	193	136	78	69	49	28
Higgs	BFS	2 480	351	149	91	65	62	56	50	45	40	41
	DFS	640	**125**	**77**	60	52	51	46	46	42	42	39
	H	2 169	239	80	**43**	**23**	**21**	**16**	**14**	**9**	**8**	**8**
Friendfeed	BFS	58 278	150	51	27	25	25	24	23	23	23	23
	DFS	13 356	803	220	82	68	68	66	58	58	59	57
	H	47 179	**10**	**4**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**

Algorithms. The core C^* can be straightforwardly found by running any of the proposed algorithms for multilayer core decomposition – BFS-ML-cores (Algorithm 3.2), DFS-ML-cores (Algorithm 3.3), or HYBRID-ML-cores (Algorithm 3.5) – and taking from the overall output core set the core maximizing the $\sigma(\cdot)$ score. However, thanks to the constraint about containment of query vertices V_Q, the various algorithms can be speeded up by preventively skipping the computation of cores that do not contain V_Q. Specifically, this corresponds to the following simple modifications:

- BFS-ML-cores (Algorithm 3.2): replace the condition at Line 7 with "**if** $V_Q \subseteq C_{\vec{k}}$ **then**".
- DFS-ML-cores (Algorithm 3.3): stop the $\vec{k}$-coresPath subroutine used at Lines 4 and 5 as soon as a core not containing V_Q is encountered and make the subroutine return only the cores containing V_Q.
- HYBRID-ML-cores (Algorithm 3.5): replace the condition at Line 9 with "**if** $V_Q \subseteq C_{\vec{k}}$ **then**".

3.6.1 Experimental results

We experimentally prove the efficiency of the modifications adopted by our algorithms for multilayer community search by reporting a comparison against the original algorithms with no such modifications. Therefore, we consider as baselines the algorithms introduced in Section 3.2 for computing the entire multilayer core decomposition, i.e., the BFS-ML-cores, DFS-ML-cores, and HYBRID-ML-cores algorithms. We vary the size $|V_Q|$ of the query-vertex set from 1 to 10. For every query-set size, we select – uniformly at random – a number of 100 different query-vertex sets from the whole vertex set. We also vary β from 0.1 and 100. The runtime with varying $|V_Q|$ is shown in Table 3.6. All results are averaged over the various query-vertex sets sampled.

In all datasets and for all algorithms, the modifications yield considerable improvement. For $|V_Q| = 1$, which is the most demanding scenario in terms of runtime, we achieve from one to three orders of magnitude of speedup in all the cases (with the exception of Amazon). As the number of query vertices increases, the modifications become even more effective: for $|V_Q| > 2$, we obtain at least one order of magnitude of speedup, up to a maximum of four orders of magnitude on the Friendfeed dataset.

As a further insight, for a number of query vertices $|V_Q| \leq 2$, the runtime of the methods for multilayer community search is strongly dependent on the underlying algorithm for multilayer core decomposition. For example, on the SacchCere and Higgs datasets, H is outperformed by BFS and DFS, respectively. The picture is instead different for $|V_Q| > 2$: H turns out to be the fastest algorithm in all the datasets, with the exception of FriendfeedTwitter, for which DFS achieves better performance up to 10 query vertices. Therefore, in general, the core-lattice visit performed by H results to be more effective in identifying the solution multilayer core quickly. In the case of FriendfeedTwitter instead, the gap between the original runtime of DFS and H is so marked that, even if H yields better speedup, it is not able to outperform DFS. This behavior is mainly motivated by the small number of layers of FriendfeedTwitter (only 2), which, as already observed beforehand, favors DFS in terms of runtime.

3.7 Summary

In this chapter we study core decomposition in multilayer networks, characterizing its usefulness, its relation to other problems, and its intrinsic complexity. We then devise three efficient algorithms for computing the whole core decomposition of a multilayer network and we show a series of non-trivial applications of the core decomposition to solve related problems. In particular:

- Given the large number of multilayer cores, we devise a recursive algorithm for efficiently computing the inner-most cores only.
- We study densest-subgraph extraction in multilayer graphs as a proper optimization problem trading off between high density and layers exhibiting high density, and show how core decomposition can be used to approximate this problem with quality guarantees.
- We show how the multilayer core-decomposition tool can be theoretical exploited to speed up the extraction of frequent cross-graph quasi-cliques, and experimentally prove the effectiveness of our approach with respect to the original algorithm for frequent cross-graph quasi-cliques.
- We generalize the multilayer community-search problem to the multilayer case and show how to exploit multilayer core decomposition to obtain optimal solutions to this problem.

Chapter 4

Core decomposition in temporal networks

A temporal network is a representation of entities (vertices), their relations (links), and how these relations are established/broken over time. Notice that here we will consider discrete times, i.e., the temporal networks can be represented as a time-ordered series of snapshots (instantaneous graphs). Extracting dense structures together with their temporal span (i.e., the period of time for which the high density is observed) is a key mining primitive to characterize such temporal networks and to identify relevant patterns. This type of pattern enables fine-grain analysis of the network dynamics and can be a building block towards more complex tasks and applications, such as finding temporally recurring subgraphs or anomalously dense ones. For instance, they can help in studying contact networks among individuals to quantify the transmission opportunities of respiratory infections in a population and uncover situations where the risk of transmission is higher, with the goal of designing mitigation strategies [107]. Anomalously dense temporal patterns among entities in a co-occurrence graph (e.g., extracted from the Twitter stream) have also been used to identify events and buzzing stories in real time [12, 44]. Another example concerns scientific collaboration and citation networks, where these patterns can help understand the dynamics of collaboration in successful professional teams, study the evolution of scientific topics, and detect emerging technologies [79].

In this chapter we adopt as a measure of density of a pattern the *minimum degree* holding among the vertices in the subgraph during the pattern's span. The problem of extracting *all* these patterns is tackled by introducing a notion of *temporal core decomposition* in which each core is associated with its *span*, i.e., an interval of *contiguous timestamps*, for which the coreness property holds. We term such a notion of temporal core *span-core*. Moreover, in several application scenarios it is typically required to identify only those dense patterns that contain a given set of query vertices. We therefore introduce the problem of *temporal community search*, whose goal is to find a set of cohesive temporal subgraphs containing the input query vertices and covering the whole temporal domain.

Challenges and contributions

As the number of possible time intervals is quadratic in the size of the input temporal domain T, the total number of span-cores is, in the worst case, quadratic in T too. The naive method to find all *span-cores*, which would be to operate a core decomposition for each of these time intervals, would therefore be very time-consuming. This is a major challenge that we tackle by deriving containment properties between span-cores and by exploiting them to devise an algorithm for computing all the *span-cores* that is significantly more efficient than the naïve exhaustive method.

We then shift our attention to the problem of finding only the *maximal span-cores*, defined as the span-cores that are not dominated by any other span-core by both the coreness property and the span. A straightforward way of approaching this problem is to filter out non-maximal span-cores during the execution of an algorithm for computing the whole span-core decomposition. However, as the maximal ones are usually much less numerous than the overall span-cores, it would be desirable to have a method that effectively exploits the maximality property and extracts maximal span-cores directly, without computing the complete decomposition. The design of an algorithm of this kind is an interesting challenge, as it contrasts with the intrinsic conceptual properties of core decomposition, based on which a core of order k can be efficiently computed from the core of order $k-1$, of which it is a subset. For this reason, at first glance, the computation of the core of the highest order would seem as hard as computing the overall core decomposition. Instead, in this work we derive a number of theoretical properties about the relationship among span-cores of different temporal intervals and, based on these findings, we show how such a challenging goal may be achieved.

Finally, we focus on the problem of *community search in temporal networks*. Community search has been extensively studied in static graphs. It requires to find a subgraph containing a given set of query vertices and maximizing a certain density measure [86, 132]. Here, we propose a formulation of the community-search problem in temporal networks as follows: given a set Q of query vertices, and a positive integer h, find a segmentation of the underlying temporal domain in h segments $\{\Delta_i\}_{i=1}^{h}$ and a subgraph S_i for every identified segment Δ_i such that each S_i contains the query vertices Q and the total density of the subgraphs is maximized. Following the bulk of the literature in community search on static networks, in our definition of temporal community search we adopt the minimum degree as a density measure.

We show that, with some manipulations, temporal community search can be reformulated as an instance of the popular *sequence segmentation* problem, which asks for partitioning a sequence of numerical values into h segments so as to minimize the sum of the penalties (according to some penalty function) on the identified segments [33]. Therefore, the classical dynamic-programming algorithm for sequence segmentation by Bellman [33] can be easily adapted to solve temporal community search in polynomial time. A criticality of this approach is that a naïve adaptation of the Bellman's algorithm takes quadratic time in the size of the input temporal domain T. As a major contribution in this regard, we prove that the set of maximal span-cores provide a sound and complete basis to still have an optimal solution to temporal community search, while at the same time leading to a significant speed-up with respect to the naïve method. In fact, let $T^* \subseteq T$ be the subset of timestamps that are covered by the span of at least one maximal span-core, together with the timestamps that immediately precede or succeed any of such spans. We show that considering T^* (instead of T) in the (adaptation of the) Bellman's algorithm is sufficient to optimally solve the underlying temporal-community-search problem instance. As, typically, $|T^*| \ll |T|$, this finding guarantees a considerable improvement in efficiency (as confirmed by our experiments).

A further challenge in our temporal-community-search problem is a typical one in community-search formulations based on minimum degree, namely, that the output subgraphs are typically large in size. We tackle this challenge by devising a method to reduce the size of the output subgraphs without affecting optimality. The proposed method is inspired by the one devised by Barbieri et al. [25] for the problem of minimum community search (in static graphs).

To summarize, the main contributions of this work are as follows:

- We introduce the notion of span-core decomposition and maximal span-core in temporal networks, characterizing structure and size of the search space and providing important containment properties (Section 4.1).

- We devise an algorithm for computing all span-cores that exploits the aforementioned containment properties and is orders of magnitude faster than a naïve method based on traditional core decomposition (Section 4.2).

- We study the problem of finding only the maximal span-cores. We derive several theoretical findings about the relationship between maximal span-cores and exploit these findings to devise an algorithm that is more efficient than computing all span-cores and discarding the non-maximal ones (Section 4.3).

- We introduce the problem of temporal community search and show how it can be solved in polynomial time via dynamic programming. We prove an important connection between temporal community search and maximal span-cores, which allows us to devise an algorithm that is considerably more efficient than the naïve dynamic-programming one. We also propose a method to achieve the critical challenge of having too large communities as output (Section 4.4).

- We provide a comprehensive experimentation on several real-world temporal networks, with millions of vertices, tens of millions of edges, and hundreds of timestamps, which attests efficiency and scalability of our methods (Section 4.5).

4.1 Temporal core decomposition

In this section we provide preliminary definitions and the needed notations, introduce the problem of finding all span-cores and only the maximal ones, and prove containment properties among span-cores that are at the basis of our efficient algorithms.

4.1.1 Span-cores

We are given a *temporal graph* $G = (V, T, \tau)$, where V is a set of vertices, $T = [0, 1, \dots, t_{max}] \subseteq \mathbb{N}$ is a discrete time domain, and $\tau : V \times V \times T \to \{0, 1\}$ is a function defining for each pair of vertices $u, v \in V$ and each timestamp $t \in T$ whether edge (u, v) exists in t. We denote $E = \{(u, v, t) \mid \tau(u, v, t) = 1\}$ the set of all temporal edges. Given a timestamp $t \in T$, $E_t = \{(u, v) \mid \tau(u, v, t) = 1\}$ is the set of edges existing at time t. A temporal interval $\Delta = [t_s, t_e]$ is contained into another temporal interval $\Delta' = [t'_s, t'_e]$, denoted $\Delta \sqsubseteq \Delta'$, if $t'_s \leq t_s$ and $t'_e \geq t_e$. Given an interval $\Delta \sqsubseteq T$, we denote $E_\Delta = \bigcap_{t \in \Delta} E_t$ the edges existing in *all timestamps* of Δ. Given a subset $S \subseteq V$ of vertices, let $E_\Delta[S] = \{(u, v) \in E_\Delta \mid u \in S, v \in S\}$ and $G_\Delta[S] = (S, E_\Delta[S])$. Finally, the *temporal degree* of a vertex u within $G_\Delta[S]$ is denoted $d_\Delta(S, u) = |\{v \in S \mid (u, v) \in E_\Delta[S]\}|$.

Definition 4.1 ((k, Δ)-core). *The* (k, Δ)-core *of a temporal graph $G = (V, T, \tau)$ is (when it exists) a maximal and non-empty set of vertices $\emptyset \neq C_{k,\Delta} \subseteq V$, such that $\forall u \in C_{k,\Delta} : d_\Delta(C_{k,\Delta}, u) \geq k$, where $\Delta \sqsubseteq T$ is a temporal interval and $k \in \mathbb{N}^+$.*

A (k, Δ)-core is thus a set of vertices implicitly defining a cohesive subgraph (where k represents the cohesiveness constraint), together with its *temporal span*, i.e., the interval Δ for which the subgraph satisfies the cohesiveness constraint. In the remainder of the chapter we refer to this type of temporal pattern as *span-core*.

The first problem we tackle in this work is to compute the *span-core decomposition* of a temporal graph G, i.e., all span-cores of G.

Problem 4.1 (Span-core decomposition). *Given a temporal graph G, find the set of all (k, Δ)-cores of G.*

Unlike standard cores of simple graphs, span-cores are not all nested into each other, due to their spans. However, they still exhibit containment properties. Indeed, it can be observed that a (k, Δ)-core is contained into any other (k', Δ')-core with less restrictive degree and span conditions, i.e., $k' \leq k$, and $\Delta' \sqsubseteq \Delta$. This property is depicted in Figure 4.1, and formally stated in the next proposition.

Proposition 4.1 (Span-core containment). *For any two span-cores $C_{k,\Delta}$, $C_{k',\Delta'}$ of a temporal graph G it holds that*

$$k' \leq k \wedge \Delta' \sqsubseteq \Delta \;\Rightarrow\; C_{k,\Delta} \subseteq C_{k',\Delta'}. \tag{4.1}$$

Proof. The result can be proved by separating the two conditions in the hypothesis, i.e., by separately showing that (*i*) $k' \leq k \Rightarrow C_{k,\Delta} \subseteq C_{k',\Delta}$, and (*ii*) $\Delta' \sqsubseteq \Delta \Rightarrow C_{k,\Delta} \subseteq C_{k,\Delta'}$. The first point holds as, keeping the span Δ fixed, the maximal set of vertices C for which $d_\Delta(C, u) \geq k$ is clearly contained in the maximal set of vertices C' for which $d_\Delta(C', u) \geq k'$, if $k' \leq k$. To prove (ii), it can be noted that $\Delta' \sqsubseteq \Delta \Rightarrow E_\Delta \subseteq E_{\Delta'}$, which implies that $\forall u \in C_{k,\Delta} : d_\Delta(C_{k,\Delta}, u) \leq d_{\Delta'}(C_{k,\Delta}, u)$. Therefore, all vertices within $C_{k,\Delta}$ satisfy the condition to be part of $C_{k,\Delta'}$ too. □

The following observation directly derives from Proposition 4.1 and states that finding all the span-cores having a fixed span Δ corresponds to computing the core decomposition of a simple graph.

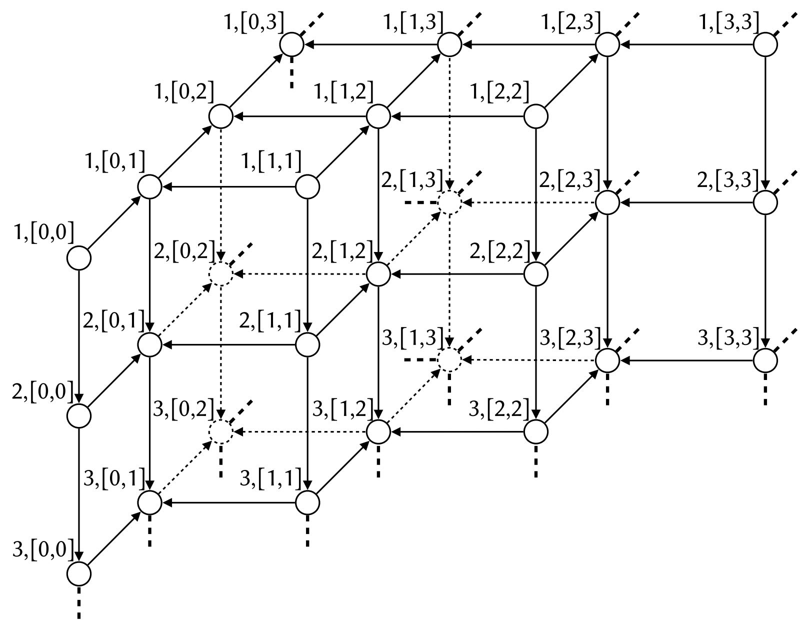


Figure 4.1: Search space: for a temporal span $\Delta = [t_s, t_e]$, the (k, Δ)-core is depicted as a node labeled "$k, [t_s, t_e]$". An arrow $C_1 \to C_2$ denotes $C_1 \supseteq C_2$ (the distinction between solid and dotted arrows is for visualization sake only).

Observation 4.1. *For a fixed temporal interval $\Delta \sqsubseteq T$, finding all span-cores that have Δ as their span is equivalent to computing the classic core decomposition [30] of the simple graph $G_\Delta = (V, E_\Delta)$.*

4.1.2 Maximal span-cores

As the total number of temporal intervals that are contained into the whole time domain T is $|T|(|T|+1)/2$, the total number of span-cores is potentially $\mathcal{O}(|T|^2 \times k_{max})$, where k_{max} is the largest value of k for which a (k, Δ)-core exists. It is thus quadratic in $|T|$, which may be too large an output for human direct inspection. In this regard, it may be useful to focus only on the most relevant cores, i.e., the *maximal* ones, as defined next.

Definition 4.2 (Maximal span-core)**.** *A span-core $C_{k,\Delta}$ of a temporal graph G is said* maximal *if there does not exist any other span-core $C_{k',\Delta'}$ of G such that $k \leq k'$ and $\Delta \sqsubseteq \Delta'$.*

Hence, a span-core is recognized as maximal if it is not dominated by another span-core both on the order k and the span Δ. Differently from the *innermost core* (i.e., the core of the highest order) in the classic core decomposition, which is unique, in our temporal setting the number of maximal span-cores is $\mathcal{O}(|T|^2)$, as, in the worst case, there may be one maximal span-core for every temporal interval. However, as observed in empirical temporal-network data, maximal span-cores are always much less than the overall span-cores: the difference is usually one order of magnitude or more. The second problem we tackle in this work is to compute the maximal span-cores of a temporal graph.

Problem 4.2 (Maximal Span-core Mining)**.** *Given a temporal graph G, find the set of all maximal (k, Δ)-cores of G.*

Clearly, one could solve Problem 4.2 by solving Problem 4.1 and filtering out all the non-maximal span-cores. However, an interesting yet challenging question is whether one can exploit the maximality condition to develop faster algorithms that can directly extract the maximal ones, without computing all the span-cores. We provide a positive answer to this question in Section 4.3.

4.2 Computing all span-cores

In this section we devise algorithms for computing a complete span-core decomposition of a temporal graph (Problem 4.1).

A naïve approach. As stated in Observation 1, for a fixed temporal interval $\Delta \sqsubseteq T$, mining all span-cores $C_{k,\Delta}$ is equivalent to computing the classic core decomposition of the graph $G_\Delta = (V, E_\Delta)$. A naïve strategy is thus to run a core-decomposition subroutine [30] on graph G_Δ for each temporal interval $\Delta \sqsubseteq T$. Such a method has time complexity $\mathcal{O}(\sum_{\Delta \sqsubseteq T}(|\Delta| \times |E|))$, i.e., $\mathcal{O}(|T|^2 \times |E|)$.

A more efficient algorithm. Looking at Figure 4.1 one can observe that the naïve algorithm only exploits one dimension of the containment property: it starts from each point on the top level, i.e., from cores of order 1, and goes down vertically with the classic core decomposition. Based on Proposition 4.1, it is possible to design a more efficient algorithm that exploits also the "horizontal containment" relationships.

Example 4.1. *Consider core $C_{1,[0,2]}$ in Figure 4.1: by Proposition 4.1 it holds that it is a subset of both $C_{1,[0,1]}$ and $C_{1,[1,2]}$. Therefore, to compute $C_{1,[0,2]}$, instead of starting from the whole V, one can start from $C_{1,[0,1]} \cap C_{1,[1,2]}$. Starting from a much smaller set of vertices can provide a substantial speed-up to the whole computation.*

This observation, although simple, produces a speed-up of orders of magnitude as we will empirically show in Section 4.5. The next straightforward corollary of Proposition 4.1 states that, not only $C_{1,[0,2]} \subseteq C_{1,[0,1]} \cap C_{1,[1,2]}$, but this is the best one can get, meaning that intersecting these two span-cores is equivalent to intersecting all span-cores structurally containing $C_{1,[0,2]}$.

Corollary 4.1. *Given a temporal graph $G = (V, T, \tau)$, and a temporal interval $\Delta = [t_s, t_e] \sqsubseteq T$, let $\Delta_+ = [\min\{t_s + 1, t_e\}, t_e]$ and $\Delta_- = [t_s, \max\{t_e - 1, t_s\}]$. It holds that*

$$C_{1,\Delta} \subseteq (C_{1,\Delta_+} \cap C_{1,\Delta_-}) = \bigcap_{\Delta' \sqsubseteq \Delta} C_{1,\Delta'}. \tag{4.2}$$

Example 4.2. *Consider again $C_{1,[0,2]}$ in Figure 4.1: Proposition 4.1 states that it is a subset of $C_{1,[0,0]}, C_{1,[0,1]}, C_{1,[1,1]}, C_{1,[1,2]}, C_{1,[2,2]}$. Corollary 4.1 suggests that there is no need to intersect them all, but only $C_{1,[0,1]}$ and $C_{1,[1,2]}$: in fact, $C_{1,[0,1]} \subseteq C_{1,[0,0]} \cap C_{1,[1,1]}$ and $C_{1,[1,2]} \subseteq C_{1,[1,1]} \cap C_{1,[2,2]}$.*

The main idea behind our efficient Span-cores algorithm (whose pseudocode is given as Algorithm 4.1) is to generate temporal intervals of increasing size (starting from size one) and, for each Δ of width larger than one, to initiate the core decomposition from $(C_{1,\Delta_+} \cap C_{1,\Delta_-})$, i.e., the smallest intersection of cores containing $C_{1,\Delta}$ (Corollary 4.1). The intervals to be processed are added to queue Q, which is initialized with the intervals of size one (Lines 2–3): these are the only intervals for which no other interval can be used to reduce the set of vertices from which the core decomposition is started, thus they have to be initialized with the whole vertex set V. The algorithm utilizes a map $\mathcal{A}$ that, given an interval Δ, returns the set of vertices to be used as a starting set of the core decomposition on Δ. The algorithm processes all intervals stored in Q, until Q has become empty (Lines 4–16). For every temporal interval Δ extracted from Q, the starting set of vertices is retrieved from $\mathcal{A}[\Delta]$ and the corresponding set of edges is identified (Line 6). Unless this is empty, the classic core-decomposition algorithm [30] is invoked over $(\mathcal{A}[\Delta], E_\Delta[\mathcal{A}[\Delta]])$ (Line 8) and its output (a set of span-cores of span Δ) is added to the ultimate output set $\mathbf{C}$ (Line 9).

Afterwards, the two intervals, denoted Δ_1 and Δ_2, for which $C_{1,\Delta}$ can be used to obtain the smallest intersections of cores containing them (Corollary 4.1) are computed at Line 10. For Δ_1 (and analogously Δ_2), we check whether $\mathcal{A}[\Delta_1]$ has already been initialized (Line 12): this would mean that previously the other "father" (i.e., smallest containing core) of C_{1,Δ_1} has been computed, thus we can intersect $C_{1,\Delta}$ with $\mathcal{A}[\Delta_1]$ and enqueue Δ_1 to be processed (Lines 13–14). Instead, if $\mathcal{A}[\Delta_1]$ was not yet initialized, we initialize it with $C_{1,\Delta}$ (Line 16): in this case Δ_1 is not enqueued because it still lacks one father to be intersected before being ready for core decomposition. This procedural update of Q ensures that both fathers of every interval in Q exist and have been previously computed, thus no a-posteriori verification is needed.

Algorithm 4.1: Span-cores

```
Input: A temporal graph $G = (V, T, \tau)$.
Output: The set $\mathbf{C}$ of all span-cores of $G$.
1  $\mathbf{C} \leftarrow \emptyset$;  $Q \leftarrow \emptyset$;  $\mathcal{A} \leftarrow \emptyset$
2  forall $t \in T$ do
3      enqueue $[t,t]$ to $Q$;  $\mathcal{A}[t,t] \leftarrow V$
4  while $Q \neq \emptyset$ do
5      dequeue $\Delta = [t_s, t_e]$ from $Q$
6      $E_\Delta[\mathcal{A}[\Delta]] \leftarrow \{(u,v) \in E_\Delta \mid u \in \mathcal{A}[\Delta], v \in \mathcal{A}[\Delta]\}$
7      if $|E_\Delta[\mathcal{A}[\Delta]]| > 0$ then
8          $\mathbf{C}_\Delta \leftarrow$ core-decomposition$(\mathcal{A}[\Delta], E_\Delta[\mathcal{A}[\Delta]])$
9          $\mathbf{C} \leftarrow \mathbf{C} \cup \mathbf{C}_\Delta$
10         $\Delta_1 = [\max\{t_s - 1, 0\}, t_e]$;  $\Delta_2 = [t_s, \min\{t_e + 1, t_{max}\}]$
11         forall $\Delta' \in \{\Delta_1, \Delta_2\} \mid \Delta' \neq \Delta$ do
12             if $\mathcal{A}[\Delta'] \neq$ NULL then
13                 $\mathcal{A}[\Delta'] \leftarrow \mathcal{A}[\Delta'] \cap C_{1,\Delta}$
14                 enqueue $\Delta'$ to $Q$
15             else
16                 $\mathcal{A}[\Delta'] \leftarrow C_{1,\Delta}$
```

Example 4.3. *Consider again the search space in Figure 4.1. Algorithm 4.1 first processes the intervals* $[0,0], [1,1], [2,2]$, *and* $[3,3]$. *Then, it intersects* $C_{1,[0,0]}$ *and* $C_{1,[1,1]}$ *to initialize* $C_{1,[0,1]}$, *intersects* $C_{1,[1,1]}$ *and* $C_{1,[2,2]}$ *to initialize* $C_{1,[1,2]}$, *and intersects* $C_{1,[2,2]}$ *and* $C_{1,[3,3]}$ *to initialize* $C_{1,[2,3]}$. *Then, it continues with the intervals of size 3: it intersects* $C_{1,[0,1]}$ *and* $C_{1,[1,2]}$ *to initialize* $C_{1,[0,2]}$ *and so on.*

The next theorem formally shows soundness and completeness of our Span-cores algorithm.

Theorem 4.1. *Algorithm 4.1 is sound and complete for Problem 4.1.*

Proof. The algorithm generates and processes a subset of temporal intervals $\mathcal{X} \subseteq \{\Delta \mid \Delta \sqsubseteq T\}$. For every interval $\Delta \subseteq \mathcal{X}$, it computes *all* span-cores $\mathbf{C}_\Delta = \{C_{1,\Delta}, C_{2,\Delta}, \ldots, C_{k_\Delta,\Delta}\}$ defined on Δ by means of the core-decomposition subroutine on the graph $(\mathcal{A}[\Delta], E_\Delta[\mathcal{A}[\Delta]])$. The set of vertices $\mathcal{A}[\Delta]$ is equivalent to $(C_{1,\Delta_+} \cap C_{1,\Delta_-})$ because of Line 13 (Corollary 4.1) and the fact that Δ is enqueued (Line 14) only when both fathers have been processed and the intersection done. The correctness of doing the classic core decomposition is guaranteed by Observation 4.1.

As for completeness, it suffices to show that the intervals $\Delta \notin \mathcal{X}$ that have not been processed by the algorithm do not yield any span-core. The algorithm generates all temporal intervals size by size, starting from those of size one and then going to larger sizes. This is done by maintaining the queue Q. As said above, an interval Δ is enqueued as soon as both C_{1,Δ_+} and C_{1,Δ_-} have been processed. Thus, an interval Δ is not in $\mathcal{X}$ only if either C_{1,Δ_+} or C_{1,Δ_-} does not exist. In this case $C_{1,\Delta}$ and all other $C_{k,\Delta}$ do not exist as well. □

Discussion. Algorithm 4.1 exploits the "horizontal containment" relationships only at the first level of the search space. For a given Δ, once the restricted starting set of vertices has been defined for $k = 1$, the traditional core decomposition is started to produce all the span-cores of span Δ. In other words, for $k > 1$ only the "vertical containment" is exploited. Consider the span-core $C_{3,[1,2]}$ in Figure 4.1: we know that it is a subset of $C_{2,[1,2]}$ ("vertical") and of $C_{3,[1,1]}$ and $C_{3,[2,2]}$ ("horizontal"). One could consider intersecting all these three span-cores before computing $C_{3,[1,2]}$. We tested this alternative approach, but concluded that the overhead of computing intersections and data-structure maintenance was outweighing the benefit of starting from a smaller vertex set.

The worst-case time complexity of Algorithm 4.1 is equal to the naïve approach, however, in practice, it is orders of magnitude faster, as shown in Section 4.5.

4.3 Computing maximal span-cores

In this section we focus on Problem 4.2: computing the *maximal* span-cores of a temporal graph.

A filtering approach. As anticipated above, a straightforward way of solving this problem consists in filtering the span-cores computed during the execution of Algorithm 4.1, so as to ultimately output only the maximal ones. This can easily be accomplished by equipping Algorithm 4.1 with a data structure $\mathcal{M}$ that stores the span-core of the highest order for every temporal interval $\Delta \sqsubseteq T$ that has been processed by the algorithm. Moreover, at the storage of a span-core $C_{k,\Delta}$ in $\mathcal{M}$, the span-cores previously stored in $\mathcal{M}$ for subintervals of the temporal interval Δ and with the same order k are removed from $\mathcal{M}$. This removal operation, together with the order in which span-cores are processed, ensures that $\mathcal{M}$ eventually contains only the maximal span-cores.

Efficient maximal-span-core finding. Our next goal is to design a more efficient algorithm that extracts maximal span-cores directly, without computing complete core decompositions, passing over more peripheral ones, and without generating all temporal cores. This is a quite challenging design principle, as it contrasts the intrinsic structural properties of core decomposition, based on which a core of order k is usually computed from the core of order $k-1$, thus making the computation of the core of the highest order as hard as computing the overall decomposition. Nevertheless, thanks to theoretical properties that relate the maximal span-cores to each other, in the temporal context such a challenge can be achieved. In the following we discuss such properties in detail, by starting from a result that has already been discussed above, but only informally.

Consider the classic core decomposition in a standard (non-temporal) graph G (Definition 2.1) and let $C_{k^*}[G]$ denote the *innermost* core of G, i.e., the non-empty k-core of G with the largest k.

Lemma 4.1. *Given a temporal graph $G = (V, T, \tau)$, let $\mathbf{C}_M$ be the set of all maximal span-cores of G, and $\mathbf{C_{inner}} = \{C_{k^*}[G_\Delta] \mid \Delta \sqsubseteq T\}$ be the set of innermost cores of all graphs G_Δ. It holds that $\mathbf{C}_M \subseteq \mathbf{C_{inner}}$.*

Proof. Every $C_{k,\Delta} \in \mathbf{C}_M$ is the innermost core of the non-temporal graph G_Δ: else, there would exist another core $C_{k',\Delta} \neq \emptyset$ with $k' > k$, implying that $C_{k,\Delta} \notin \mathbf{C}_M$. □

Lemma 4.1 states that each maximal span-core is an innermost core of a G_Δ, for some temporal interval $\Delta \sqsubseteq T$. Hence, there can exist at most one maximal span-core for every $\Delta \sqsubseteq T$ (while an interval Δ may not yield any maximal span-core). The key question to design an efficient maximal-span-core-mining algorithm thus becomes how to extract innermost cores of the graphs G_Δ more efficiently than by computing the full core decompositions of all G_Δ. The answer to this question comes from the result stated in the next two lemmas (with Lemma 4.2 being auxiliary to Lemma 4.3).

Lemma 4.2. *Given a temporal graph $G = (V, T, \tau)$, and three temporal intervals $\Delta = [t_s, t_e] \sqsubseteq T$, $\Delta' = [t_s{-}1, t_e] \sqsubseteq T$, and $\Delta'' = [t_s, t_e{+}1] \sqsubseteq T$. The innermost core $C_{k^*}[G_\Delta]$ is a maximal span-core of G if and only if $k^* > \max\{k', k''\}$ where k' and k'' are the orders of the innermost cores of $G_{\Delta'}$ and $G_{\Delta''}$, respectively.*

Proof. The "⇒" part comes directly from the definition of maximal span-core (Definition 4.2): if k^* were not larger than $\max\{k', k''\}$, then $C_{k^*}[G_\Delta]$ would be dominated by another span-core both on the order and on the span (as both Δ' and Δ'' are superintervals of Δ). For the "⇐" part, from Lemma 4.1 and Proposition 4.1 it follows that $\max\{k', k''\}$ is an upper bound on the maximum order of a span-core of a superinterval of Δ. Therefore, $k^* > \max\{k', k''\}$ implies that there cannot exist any other span-core that dominates $C_{k^*}[G_\Delta]$ both on the order and on the span. □

Lemma 4.3. *Given G, Δ, Δ', Δ'', k', and k'' defined as in Lemma 4.2, let $\widetilde{V} = \{u \in V \mid d_\Delta(V, u) > \max\{k', k''\}\}$, and let $C_{k^*}[G_\Delta[\widetilde{V}]]$ be the innermost core of $G_\Delta[\widetilde{V}]$. If $k^* > \max\{k', k''\}$, then $C_{k^*}[G_\Delta[\widetilde{V}]]$ is a maximal span-core; otherwise, no maximal span-core exists for Δ.*

Proof. Lemma 4.2 states that, to be recognized as a maximal span-core, the innermost core of G_Δ should have order larger than $\max\{k', k''\}$. This means that, if the innermost core of G_Δ is a maximal span-core, all vertices $u \notin \widetilde{V}$ cannot be part of it. Therefore, G_Δ yields a maximal span-core only if the innermost core of subgraph $G_\Delta[\widetilde{V}]$ has order $k^* > \max\{k', k''\}$. □

Algorithm 4.2: Maximal-span-cores

```
Input: A temporal graph $G = (V, T, \tau)$.
Output: The set $\mathbf{C}_M$ of all maximal span-cores of $G$.
1  $\mathbf{C}_M \leftarrow \emptyset$
2  $\mathcal{K}'[t] \leftarrow 0, \forall t \in T$
3  forall $t_s \in [0, 1, \ldots, t_{max}]$ do
4      $t^* \leftarrow \max\{t_e \in [t_s, t_{max}] \mid E_{[t_s,t_e]} \neq \emptyset\}$
5      $k'' \leftarrow 0$
6      forall $t_e \in [t^*, t^*-1, \ldots, t_s]$ do
7          $\Delta \leftarrow [t_s, t_e]$
8          $lb \leftarrow \max\{\mathcal{K}'[t_e], k''\}$
9          $V_{lb} \leftarrow \{u \in V \mid d_\Delta(V, u) > lb\}$
10         $E_\Delta[V_{lb}] \leftarrow \{(u, v) \in E_\Delta \mid u \in V_{lb}, v \in V_{lb}\}$
11         $C \leftarrow$ innermost-core$(V_{lb}, E_\Delta[V_{lb}])$
12         $k^* \leftarrow$ order of $C$
13         if $k^* > lb$ then
14             $\mathbf{C}_M \leftarrow \mathbf{C}_M \cup \{C\}$
15         $k'' \leftarrow \max\{k'', k^*\}$;  $\mathcal{K}'[t_e] \leftarrow \max\{\mathcal{K}'[t_e], k''\}$
```

Lemma 4.3 provides the basis of our efficient method for extracting maximal span-cores. Basically, it states that, to verify whether a certain temporal interval $\Delta = [t_s, t_e]$ yields a maximal span-core (and, if so, compute it), there is no need to consider the whole graph G_Δ, rather it suffices to start from a smaller subgraph, which is given by all vertices whose temporal degree is larger than the maximum between the orders of the innermost cores of intervals $\Delta' = [t_s-1, t_e]$ and $\Delta'' = [t_s, t_e+1]$. This finding suggests a strategy that is opposite to the one used for computing the overall span-core decomposition: a *top-down* strategy that processes temporal intervals starting from the larger ones. Indeed, in addition to exploiting the result in Lemma 4.3, this way of exploring the temporal-interval space allows us to skip the computation of complete core decompositions of the whole "singleton-interval" graphs $\{G_{[t,t]}\}_{t \in T}$, which may easily become a critical bottleneck, as they are the largest ones among the graphs induced by temporal intervals.

The Maximal-span-cores algorithm. Algorithm 4.2 iterates over all timestamps $t_s \in T$ in *increasing order* (Line 3), and for each t_s it first finds all the maximal span-cores that have span starting in t_s. This way of proceeding *ensures that a span-core that is recognized as maximal will not be later dominated by another span-core.* Indeed, an interval $[t_s, t_e]$ can never be contained in another interval $[t'_s, t'_e]$ with $t_s < t'_s$. For a given t_s, all maximal span-cores are computed as follows. First, the maximum timestamp $\geq t_s$ such that the corresponding edge set $E_{[t_s,t_e]}$ is not empty is identified as t^* (Line 4). Then, all intervals $\Delta = [t_s, t_e]$ are considered one by one in *decreasing order* of t_e (Lines 6–7): this again *guarantees that a span-core that is recognized as maximal will not be later dominated by another span-core, as the intervals are processed from the largest to the smallest.* At each iteration of the internal cycle, the algorithm resorts to Lemma 4.3 and computes the lower bound lb on the order of the innermost core of G_Δ to be recognized as maximal, by taking the maximum between $\mathcal{K}'[t_e]$ and k'' (Line 8). $\mathcal{K}'$ is a map that maintains, for every timestamp $t \in [t_s, t^*]$, the order of the innermost core of graph $G_{\Delta'}$, where $\Delta' = [t_s-1, t]$ (i.e., $\mathcal{K}'[t]$ stores what in Lemmas 4.2–4.3 is denoted as k'). Whereas k'' stores the order of the innermost core of $G_{\Delta''}$, where $\Delta'' = [t_s, t_e+1]$. Afterwards, the sets of vertices V_{lb} and of edges $E_\Delta[V_{lb}]$ that comply with this lower-bound constraint are built (Lines 9–10), and the innermost core of the subgraph $(V_{lb}, E_\Delta[V_{lb}])$ is extracted (Lines 11–12). Ultimately, based again on Lemma 4.3, such a core is added to the output set of maximal span-cores only if its order is actually larger than lb (Lines 13–14), and the values of k'' and $\mathcal{K}'[t_e]$ are updated (Line 15). Specifically, note that the order k^* of core C may in principle be less than k'', as C is extracted from a subgraph of G_Δ. If this happens, it means that the actual order of the innermost core of G_Δ is equal to k''. This motivates the update rules (and their order) reported in Line 15.

Theorem 4.2. *Algorithm 4.2 is sound and complete for Problem 4.2.*

Proof. The algorithm processes all temporal intervals $\Delta \sqsubseteq T$ yielding a non-empty edge set E_Δ, in an order such that no interval is processed before one of its superintervals: this guarantees that a span-core recognized as maximal will not be dominated by another span-core found later on. For every Δ it extracts a core C that is used as a proxy of the innermost core of graph G_Δ. C is added to the output set $\mathbf{C}_M$ only if Lemma 4.3 recognizes it as a maximal span-core, otherwise it is discarded. This proves the soundness of the algorithm. Completeness follows from Lemma 4.1, which states that to extract all maximal span-cores it suffices to focus on the innermost cores of graphs $\{G_\Delta \mid \Delta \sqsubseteq T\}$, and Lemma 4.3 again, which states the condition for a proxy core C to be safely discarded because it is a non-maximal span-core. □

Discussion. The worst-case time complexity of Algorithm 4.2 is the same as the algorithm for computing the overall span-core decomposition, i.e., $\mathcal{O}(|T|^2 \times |E|)$. It is worth mentioning that it is not possible to do better than this, as the output itself is potentially quadratic in $|T|$. However, as we will show in Section 4.5, the proposed algorithm is in practice much more efficient than computing the overall span-core decomposition and filtering out the non-maximal span-cores as, in this case, we avoid the visit of portions of the span-core search space and the computations are run over subgraphs of reduced dimensions.

To conclude, we discuss how the crucial operation of building the subgraph $(V_{lb}, E_\Delta[V_{lb}])$ may be carried out efficiently in terms of both time and space. Consider a fixed timestamp $t_s \in [0, \ldots, t_{max}]$. The following reasoning holds for every t_s. Let $E^-(t_e) = E_{[t_s,t_e]} \setminus E_{[t_s,t_e+1]}$ be the set of edges that are in $E_{[t_s,t_e]}$ but not in $E_{[t_s,t_e+1]}$, for $t_e \in [t_s, \ldots, t^* - 1]$. As a first general step, for each t_s, we compute and store *all* edge sets $\{E^-(t_e)\}_{t_e \in [t_s, t^*-1]}$. These operations can be accomplished in $\mathcal{O}(|T| \times |E|)$ overall time, because every $E^-(t_e)$ can be computed incrementally from $E_{[t_s,t_e]}$ as $E^-(t_e) = \{(u,v) \in E_{[t_s,t_e]} \mid \tau(u,v,t_e+1) = 0\}$. Moreover, for any timestamp t_e, we keep a map $\mathcal{D}$ storing all vertices of $G_{[t_s,t_e]}$ organized by degree. Specifically, the set $\mathcal{D}[k]$ contains all vertices having degree $> k$ in $G_{[t_s,t_e]}$. Every vertex in $\mathcal{D}$ is thus replicated a number of times equal to its degree. This way, the overall space taken by $\mathcal{D}$ is $\mathcal{O}(|E|)$, i.e., as much space as G. $\mathcal{D}$ is initialized as empty (when $t_e = t^*$) and repeatedly augmented as t_e decreases, by a linear scan of the various $E^-(t_e)$. The overall filling of $\mathcal{D}$ (for all t_e) therefore takes $\mathcal{O}(|T| \times |E|)$ time. Then, the desired V_{lb} can be computed in constant time simply as $V_{lb} = \mathcal{D}[lb]$.

As for $E_\Delta[V_{lb}]$, for any t_e, we first reconstruct $E_{[t_s,t_e]}$ as $E_{[t_s,t_e+1]} \cup E^-(t_e)$, having previously computed $E_{[t_s,t_e+1]}$. Note that storing all $E^-(t_e)$ takes $\mathcal{O}(|E|)$ space. That is why we store all $E^-(t_e)$ and reconstruct $E_{[t_s,t_e]}$ afterward (instead of storing the latter, which would take $\mathcal{O}(|T| \times |E|)$ space). $E_\Delta[V_{lb}]$ is ultimately derived by a linear scan of $E_{[t_s,t_e]}$, taking all edges in $E_{[t_s,t_e]}$ having both endpoints in V_{lb}. This way, the step of building $E_\Delta[V_{lb}]$ for all t_e takes again $\mathcal{O}(|T| \times |E|)$ overall time.

4.4 Temporal community search

Community search in static graphs aims at finding a dense subgraph (community) containing a set of input query vertices [86, 132]. In the temporal setting it is very likely that the communities spanning the query vertices change over time. To be more precise, it may happen that a certain subgraph S is a well-representative community for the given query vertices Q, but only for a certain time interval Δ. Instead, for another time interval Δ', a relevant community for Q might correspond to a completely different subgraph S'. For this reason, we formulate community search on temporal networks as the problem of finding h subgraphs (with $h > 0$ being an input parameter) containing the query vertices, together with their temporal span, such that the sum of the density of those subgraphs is maximized and the union of their temporal spans corresponds to the whole input temporal domain. Among the many densities proposed in the literature, here we follow the seminal work by Sozio and Gionis [215] on community search and adopt the minimum degree. Formally:

Problem 4.3 (Temporal Community Search). *Given a temporal graph $G = (V, T, \tau)$, a set $Q \subseteq V$ of query vertices, and a positive integer $h \in \mathbb{N}^+$, find a set $\{\langle S_i, \Delta_i \rangle\}_{i=1}^h$ of h pairs such* *that (i) $\forall 1 \leq i \leq h : Q \subseteq S_i \subseteq V$, (ii) $\bigcup_{1 \leq i \leq h} \Delta_i = T$, and (iii) the following is maximized:*

$$\sum_{i=1}^{h} \min_{u \in S_i} d_{\Delta_i}(S_i, u). \tag{4.3}$$

The input integer h is a user-defined parameter that gives the analyst the flexibility of requiring a specific number of output temporal communities, which might vary from application to application.

4.4.1 Connection with sequence segmentation

Here we provide some theoretical insights into the Temporal Community Search problem. The main result we provide at the end of this subsection is an interesting connection with the well-established Sequence Segmentation problem [33]. As shown in the next subsections, such a result forms the basis for algorithmic design.

Let us first consider a single-interval variant of Problem 4.3: for a fixed temporal interval Δ, find a subgraph containing the input set Q of query vertices that maximizes the minimum temporal degree within Δ. Formally:

Problem 4.4 (Single Temporal Community Search)**.** *Given a temporal graph $G = (V, T, \tau)$, a set $Q \subseteq V$ of query vertices, and an interval $\Delta \sqsubseteq T$, find*

$$S^* = \text{argmax}_{Q \subseteq S \subseteq V} \min_{u \in S} d_{\Delta}(S, u). \tag{4.4}$$

It is easy to see that solving Problem 4.4 corresponds to solving minimum-degree-based community search on graph G_Δ. Therefore, a solution to Problem 4.4 can straightforwardly be computed by applying a standard result on minimum-degree-based community search, which states that the highest-order core containing all query vertices is a solution to that problem [25]. This finding is formalized next.

Definition 4.3 ((Q, Δ)-highest-order-span-core)**.** *Given a temporal graph $G = (V, T, \tau)$, a set $Q \subseteq V$ of query vertices, and an interval $\Delta \sqsubseteq T$, the* (Q, Δ)-highest-order-span-core *of G, denoted $C^*_{Q,\Delta}$, is defined as the highest-order span-core among all span-cores of G with temporal span Δ and containing all query vertices in Q. Let also $v^*_{Q,\Delta}$ denote the order of $C^*_{Q,\Delta}$.*

Fact 4.1. *Given a temporal graph $G = (V, T, \tau)$, a set $Q \subseteq V$ of query vertices, and an interval $\Delta \sqsubseteq T$, the (Q, Δ)-highest-order-span-core of G is a solution to Problem 4.4 on input $\langle G, Q, \Delta \rangle$.*

Note that Problem 4.4 may have multiple solutions: $C^*_{Q,\Delta}$ is only one of those possibly many ones. $C^*_{Q,\Delta}$ can be computed by running a core decomposition on (static) graph G_Δ, and stopping it when the first core that does not contain all query vertices in Q has been encountered. Therefore, Problem 4.4 can be solved in $\mathcal{O}(|\Delta| \times |E|)$ time.

In light of the above findings, an alternative yet equivalent way of formulating our Temporal Community Search problem is to ask for a *segmentation* (i.e., a partition) of the time domain T into a set $\{\Delta_i\}_{i=1}^h$ of h intervals so as to maximize the sum $\sum_{i=1}^h v^*_{Q,\Delta_i}$ of the orders of the (Q, Δ)-highest-order-span-cores of those identified intervals. Once such an optimal segmentation of T has been computed, the ultimate $\{\langle S_i, \Delta_i \rangle\}_{i=1}^h$ pairs are derived by simply setting $S_i = C^*_{Q,\Delta_i}$, $\forall 1 \leq i \leq h$. Formally:

Problem 4.5 (Alternative formulation of Problem 4.3)**.** *Given a temporal graph $G = (V, T, \tau)$, a set $Q \subseteq V$ of query vertices, and a positive integer $h \in \mathbb{N}^+$, find a set $\{\langle S_i, \Delta_i \rangle\}_{i=1}^h$ of h pairs such that (i) $\forall 1 \leq i \leq h : S_i = C^*_{Q,\Delta_i}$, (ii) $\{\Delta_i\}_{i=1}^h$ is a partition of T, and (iii) the following is maximized:*

$$\sum_{i=1}^{h} v^*_{Q,\Delta_i}. \tag{4.5}$$

Algorithm 4.3: Temporal-community-search

```
Input: A temporal graph $G = (V, E, T)$, a set $Q \subseteq V$ of query vertices, an integer $h \in \mathbb{N}^+$.
Output: A set $\{\langle S_i, \Delta_i \rangle\}_{i=1}^h$, where $Q \subseteq S_i \subseteq V$, $\forall 1 \leq i \leq h$, and $\{\Delta_i\}_{i=1}^h$ is a partition
        of $T$.
/* Initialization                                                          */
1  Compute $v^*_{Q,\Delta}$ and $C^*_{Q,\Delta}$, $\forall \Delta \sqsubseteq T$, via $Q$-constrained span-core decomposition
2  $\mathbf{P} \leftarrow$ an empty $(|T| \times h)$-dimensional matrix                 // Penalty matrix
3  $\mathbf{R} \leftarrow$ an empty $(|T| \times h)$-dimensional matrix          // Reconstruction matrix
4  forall $t \in T$ do
5      $\mathbf{P}[t, 0] \leftarrow -v^*_{Q,[0,t]}$
6      $\mathbf{R}[t, 0] \leftarrow 0$
/* Dynamic-programming step                                                */
7  forall $t \in T$ do
8      forall $i \in [1, h)$ do
9          $\mathbf{P}[t, i] \leftarrow \min_{\ell \in [0,t]} \mathbf{P}[\ell, i-1] - v^*_{Q,[\ell+1,t]}$
10         $\mathbf{R}[t, i] \leftarrow \mathrm{argmin}_{\ell \in [0,t]} \mathbf{P}[\ell, i-1] - v^*_{Q,[\ell+1,t]}$
/* Reconstruction of the solution                                          */
11 $ub \leftarrow t_{max}$
12 forall $i \in (h, 0]$ do
13     $lb \leftarrow \mathbf{R}[ub, i]$
14     $\Delta_i \leftarrow [lb, ub]$
15     $ub \leftarrow lb - 1$
16 forall $i \in (h, 0]$ do
17     $S_i \leftarrow C^*_{Q,\Delta_i}$
```

Correspondence between Problem 4.3 and Problem 4.5 easily follows from Fact 4.1 and from the observation that for any feasible solution $\{\langle S_i, \Delta_i \rangle\}_{i=1}^h$ to Problem 4.3 with overlapping intervals, there exists an overlapping-interval-free feasible solution with not smaller objective-function value. To see the latter, for any two overlapping intervals Δ_i and Δ_j, simply replace one of the two intervals, say Δ_i, with $\Delta'_i = \Delta_i \setminus (\Delta_i \cap \Delta_j)$. As $\Delta'_i \sqsubseteq \Delta_i$, it holds that $v^*_{Q,\Delta'_i} \geq v^*_{Q,\Delta_i}$, therefore the resulting overlapping-interval-free solution will have objective-function value greater than or equal to the objective-function value of the starting solution with overlapping intervals.

Thanks to the reformulation in Problem 4.5, it is immediate to observe that our Temporal Community Search problem is an instance of the well-established Sequence Segmentation problem, which asks for partitioning a sequence of numerical values into b segments so as to minimize the sum of the penalties (according to some penalty function) on each identified segment [33]:

Problem 4.6 (Sequence Segmentation [33]). *Given a sequence $X = (x_0, x_1, \ldots, x_{max})$ of numerical values, and a function $p : \{Y\}_{Y \sqsubseteq X} \to \mathbb{R}$ that assigns a penalty score to every subsequence Y of X, partition X into a set $\{X_i\}_{i=1}^b$ of b subsequences such that $\sum_{i=1}^b p(X_i)$ is minimized.*

Fact 4.2. Temporal Community Search *(Problem 4.3) on input $\langle G = (V, T, \tau), Q, h \rangle$ is an instance of* Sequence Segmentation *(Problem 4.6) with $X = T$, $b = h$, and $\forall \Delta \sqsubseteq T : p(\Delta) = -v^*_{Q,\Delta}$.*

In the following two subsections we show how to exploit the result in Fact 4.2 (and a further important finding about maximal span-cores) to design efficient algorithms for our Temporal Community Search problem.

4.4.2 A basic algorithm (based on all span-cores)

Sequence Segmentation can be solved in $\mathcal{O}(|X|^2 \times h + \tau_p)$ time via dynamic programming [33], where τ_p is the overall time spent for computing the penalty score of all subsequences of the input sequence X (according to the given penalty function p). Thanks to the connection shown in Fact 4.2, the dynamic-programming algorithm for Sequence Segmentation can be easily adapted to solve Temporal Community Search as well. The pseudocode of this algorithm – termed Temporal-community-search – is reported as Algorithm 4.3, and described next.

The Temporal-community-search algorithm makes use of two $(|T| \times h)$-dimensional matrices, i.e., $\mathbf{P}$ and $\mathbf{R}$. Matrix $\mathbf{P}$ represents the *penalty matrix*. It contains, $\forall t \in T$, $\forall i \in [0, h)$, the minimum cost of segmenting the sequence corresponding to the first t timestamps of T into $i+1$ segments. As a result, $\mathbf{P}[t_{max}, h-1]$ contains the objective-function value of the ultimate optimal solution to Problem 4.5. Matrix $\mathbf{R}$ is the *reconstruction matrix*. It provides information about the optimal segmentation, and is used at the end of the algorithm to reconstruct the output $\{\Delta_i\}_{i=1}^{h}$. Note that the algorithm does not explicitly compute the S_i subgraphs corresponding to the optimal Δ_i intervals. In fact, as discussed above, each S_i can be easily retrieved at the end of the algorithm, by simply setting it equal to the corresponding (Q, Δ_i)-highest-order-span-core C^*_{Q,Δ_i}. According to Fact 4.2, the penalty score of an interval $\Delta \sqsubseteq T$ corresponds to $-v^*_{Q,\Delta}$, i.e., the negative of the order of the (Q, Δ)-highest-order-span-core $C^*_{Q,\Delta}$. All individual $v^*_{Q,\Delta}$ values, for all $\Delta \sqsubseteq T$, are efficiently computed altogether, at the beginning of the algorithm, via a "Q-constrained" variant of span-core decomposition (an alternative, but much less efficient strategy consists in computing every single $v^*_{Q,\Delta}$ from scratch, on the fly). Specifically, a simple (yet more efficient) variant of the span-core decomposition algorithm (Algorithm 4.1) is employed for this purpose, which outputs only those span-cores containing all the vertices in Q. This is easily achievable by stopping the core-decomposition subroutine, for every interval $\Delta \sqsubseteq T$, as soon as a core not containing all query vertices in Q has been encountered.

The time complexity of Algorithm 4.3 is $\mathcal{O}(|T|^2 \times h + \tau_{sc})$, where τ_{sc} is the time spent for computing the Q-constrained span-core decomposition of the input graph G.

4.4.3 A more efficient algorithm (based on maximal span-cores)

A more efficient algorithm can be designed by noticing that, actually, one does not need to consider all timestamps in T in the dynamic-programming step. Rather, focusing on a subset $T^* \subseteq T$ – which is properly defined based on the maximal span-cores of the input graph, see next – allows for significantly reducing the dimensionality of the penalty matrix P and the reconstruction matrix R, hence the overall time complexity of the algorithm, without affecting optimality of the output solution. The following fact provides the theoretical basis for defining such a reduced temporal domain T^*.

Fact 4.3. *Given a temporal graph $G = (V, T, \tau)$ and a set $Q \subseteq V$ of query vertices, let $\mathbf{C}_M(Q)$ be the set of all Q-constrained maximal span-cores of G. For a temporal interval $\Delta \sqsubseteq T$, it holds that $v^*_{Q,\Delta} = \max\{0, \max\{k \mid C_{k,\Delta'} \in \mathbf{C}_M(Q), \Delta \sqsubseteq \Delta'\}\}$.*

Fact 4.3 states that the penalty score $v^*_{Q,\Delta}$ of an interval Δ corresponds to the maximum among the orders of the Q-constrained maximal span-cores whose span includes Δ, if some exist. If an interval Δ is not a subset of any span of a Q-constrained maximal span-core, then $v^*_{Q,\Delta} = 0$. In that case, therefore, Δ can be safely discarded, as it cannot be part of the optimal solution of the given Temporal Community Search problem instance (unless it is needed to fill possible "holes", see below). The ultimate consequence of this finding is that the aforementioned reduced temporal domain T^* is identified by the timestamps covered by the spans of the maximal span-cores, along with auxiliary timestamps, which are needed to ensure a smooth execution of the dynamic-programming step, as well as a correct handling of some extreme cases. Specifically, let $\mathbf{D} = \{\Delta \sqsubseteq T \mid C_{k,\Delta} \in C_M(Q)\}$ be the set of the spans of the Q-constrained maximal span-cores of the input graph, and $T_{\mathbf{D}} = \bigcup_{\Delta \in \mathbf{D}} \Delta$ be the set of timestamps that are part of a span of a Q-constrained maximal span-core. The first two sets of auxiliary timestamps correspond to the timestamps that immediately precede and succeed the intervals in $\mathbf{D}$, i.e., the sets $T^+_{\mathbf{D}} = \{\min\{t_e + 1, t_{max}\} \mid [t_s, t_e] \in \mathbf{D}\}$ and $T^-_{\mathbf{D}} = \{\max\{t_s - 1, 0\} \mid [t_s, t_e] \in \mathbf{D}\}$, respectively. The timestamps in $T^+_{\mathbf{D}}$ and $T^-_{\mathbf{D}}$ (along with the last timestamp t_{max} of the input temporal domain T) are needed to allow the dynamic-programming step to identify a solution that actually covers the whole temporal domain T (as per Condition (*ii*) of Problem 4.3). In particular, such timestamps may be interpreted as a trick to give the dynamic-programming step the flexibility to select "holes"

Algorithm 4.4: Efficient-temporal-community-search

```
Input: A temporal graph $G = (V, E, T)$, a set $Q \subseteq V$ of query vertices, an integer $h \in \mathbb{N}^+$.
Output: A set $\{\langle S_i, \Delta_i \rangle\}_{i=1}^h$, where $Q \subseteq S_i \subseteq V$, $\forall 1 \leq i \leq h$, and $\{\Delta_i\}_{i=1}^h$ is a partition
        of $T$.
/* Identification of $T^*$                                                         */
1  Compute the set $\mathbf{C}_M(Q)$ of $Q$-constrained maximal span-cores of $G$
2  $\mathbf{D} \leftarrow \{\Delta \sqsubseteq T \mid C_{k,\Delta} \in \mathbf{C}_M(Q)\}$
3  $T_\mathbf{D} \leftarrow \bigcup_{\Delta \in \mathbf{D}} \Delta$;  $T_\mathbf{D}^+ \leftarrow \{\min\{t_e+1, t_{max}\} \mid [t_s, t_e] \in \mathbf{D}\}$;
   $T_\mathbf{D}^- \leftarrow \{\max\{t_s-1, 0\} \mid [t_s, t_e] \in \mathbf{D}\}$
4  $T_{sup} \leftarrow \{t_i \in T \setminus (T_\mathbf{D} \cup T_\mathbf{D}^- \cup T_\mathbf{D}^+ \cup \{t_{max}\}) \mid i \in [1, h+1-|T_\mathbf{D} \cup T_\mathbf{D}^- \cup T_\mathbf{D}^+ \cup \{t_{max}\}|]\}$
5  $T^* \leftarrow T_\mathbf{D} \cup T_\mathbf{D}^+ \cup T_\mathbf{D}^- \cup \{t_{max}\} \cup T_{sup}$
/* Initialization                                                                   */
6  Compute $v^*_{Q,\Delta}$, $\forall \Delta \sqsubseteq T$
7  $\mathbf{M} \leftarrow$ mapping function $[0, |T^*|) \to T^*$
8  $\mathbf{P} \leftarrow$ an empty $(|T^*| \times h)$-dimensional matrix              // Penalty matrix
9  $\mathbf{R} \leftarrow$ an empty $(|T^*| \times h)$-dimensional matrix       // Reconstruction matrix
10 forall $r \in [0, |T^*|)$ do
11     $\mathbf{P}[r, 0] \leftarrow -v^*_{Q,[0,\mathbf{M}[r]]}$
12     $\mathbf{R}[r, 0] \leftarrow 0$
/* Dynamic-programming step                                                         */
13 forall $r \in [0, |T^*|)$ do
14     forall $i \in [1, h)$ do
15         $\mathbf{P}[r, i] \leftarrow \min_{\ell \in [0,r]} \mathbf{P}[\ell, i-1] - v^*_{Q,[\mathbf{M}[\ell+1],\mathbf{M}[r]]}$
16         $\mathbf{R}[r, i] \leftarrow \text{argmin}_{\ell \in [0,r]} \mathbf{P}[\ell, i-1] - v^*_{Q,[\mathbf{M}[\ell+1],\mathbf{M}[r]]}$
/* Reconstruction of the solution                                                   */
17 $ub \leftarrow |T^*| - 1$
18 forall $i \in (h, 0]$ do
19     $lb \leftarrow \mathbf{R}[ub, i]$
20     $\Delta_i \leftarrow [\mathbf{M}[lb], \mathbf{M}[ub]]$
21     $ub \leftarrow lb - 1$
22 forall $i \in (h, 0]$ do
23     $S_i \leftarrow C^*_{Q,\Delta_i}$
```

(i.e., time intervals in-between two consecutive but not necessarily contiguous timestamps in $T_\mathbf{D}$). Moreover, we define T_{sup} as the set of the first $h + 1 - |T_\mathbf{D} \cup T_\mathbf{D}^- \cup T_\mathbf{D}^+ \cup \{t_{max}\}|$ timestamps of T not contained in $T_\mathbf{D} \cup T_\mathbf{D}^- \cup T_\mathbf{D}^+ \cup \{t_{max}\}$, i.e., $T_{sup} = \{t_i \in T \setminus (T_\mathbf{D} \cup T_\mathbf{D}^- \cup T_\mathbf{D}^+ \cup \{t_{max}\}) \mid i \in [1, h+1-|T_\mathbf{D} \cup T_\mathbf{D}^- \cup T_\mathbf{D}^+ \cup \{t_{max}\}|]\}$. The timestamps in T_{sup} are further auxiliary timestamps that are needed to return a correct h-sized solution when the timestamps in $T_\mathbf{D} \cup T_\mathbf{D}^- \cup T_\mathbf{D}^+ \cup \{t_{max}\}$ are less than $h + 1$ (the minimum number of timestamps required in T^* to have a solution of size h). Note that T_{sup} is nonempty only if $|T_\mathbf{D} \cup T_\mathbf{D}^- \cup T_\mathbf{D}^+ \cup \{t_{max}\}| < h+1$. Ultimately, T^* is defined as

$$T^* = T_\mathbf{D} \cup T_\mathbf{D}^+ \cup T_\mathbf{D}^- \cup \{t_{max}\} \cup T_{sup}. \tag{4.6}$$

The proposed more efficient method for Temporal Community Search, termed Efficient-temporal-community-search, is summarized in Algorithm 4.4 and described next. The first five lines of the algorithm are devoted to the identification of T^*. As said above, matrices $\mathbf{P}$ and $\mathbf{R}$ have here reduced dimensionality with respect to Algorithm 4.3: they are $(|T^*| \times h)$-dimensional matrices, where $|T^*| \leq |T|$. A mapping function $\mathbf{M}$ is used to assign an index within $[0, |T^*|)$ to every timestamp in $|T^*|$ (Line 6). Such a mapping is needed to have every timestamp in $|T^*|$ logically assigned to a row of matrices $\mathbf{P}$ and $\mathbf{R}$. The rest of the algorithm resembles Algorithm 4.3, except for the fact that $\mathbf{M}$ is used every time that a row index has to be mapped to its corresponding timestamp (e.g., during the reconstruction of the solution).

An important point to clarify is that, during the execution of the Efficient-temporal-community-search algorithm, we might need the penalty score $v^*_{Q,\Delta}$ of intervals $\Delta \sqsubseteq T$ corresponding to *non-maximal* (Q-constrained) span-cores. Therefore, the algorithm needs the $v^*_{Q,\Delta}$ score of *all* intervals $\Delta \sqsubseteq T$. To compute these $v^*_{Q,\Delta}$ scores (and, related to this, the set $\mathbf{C}_M(Q)$ of Q-constrained maximal span-cores, at Line 1), there are two main options. The first one consists in computing the whole Q-constrained span-core decomposition (as done in Algorithm 4.3), keep the $v^*_{Q,\Delta}$ scores of all such cores, and eventually compute $\mathbf{C}_M(Q)$ by simply filtering out non-maximal span-cores. The second option corresponds instead to compute $\mathbf{C}_M(Q)$ directly, without passing through the whole Q-constrained span-core decomposition. This may be carried out by running a simple variant of the algorithm for computing maximal span-cores (Algorithm 4.2), where containment of query vertices is added as a further constraint. The computation of all the $v^*_{Q,\Delta}$ scores comes for free during the execution of this algorithm for Q-constrained maximal span-cores: these scores can therefore be retained by adding a few straightforward (constant-time) instructions to that algorithm. In our implementation we stick to the latter, as the Maximal-span-cores algorithm has been experimentally recognized as faster than the naïve filtering approach in all tested datasets.

The time complexity of the proposed Efficient-temporal-community-search algorithm is $\mathcal{O}(|T^*|^2 \times h + \tau_{msc})$, with τ_{msc} being the time spent in computing the Q-constrained maximal span-cores and the penalty scores $v^*_{Q,\Delta}$. As in practice (attested by our experiments) $|T^*| \ll |T|$, the proposed Efficient-temporal-community-search algorithm is expected to be much more efficient than its naïve counterpart, i.e., Algorithm 4.3.

4.4.4 Minimum community search

An instance of Temporal Community Search may admit several optimal solutions which might differ either in terms of output intervals $\{\Delta_i\}_{i=1}^h$, or in terms of subgraphs assigned to the various identified intervals. More precisely, the latter refers to the fact that two optimal solutions might find the same segmentation $\{\Delta_i\}_{i=1}^h$ of the input temporal domain, but select different subgraphs S_i for any interval Δ_i. Therefore, if the communities S_i are not chosen carefully, they may result to be excessively large, not really cohesive, and containing redundant/outlying vertices. This is a well-recognized issue of minimum-degree-based community search [215]. At the same time, large communities might include more cohesive and denser subgraphs that still exhibit optimality. Motivated by this, in this subsection we devise a method to refine the communities originally found by our algorithms for Temporal Community Search, specifically attempting to minimize their size while preserving optimality. The main idea behind our refinement method is based on the following result:

Proposition 4.2 (Community containment). *Given a temporal graph $G = (V, T, \tau)$, a set $Q \subseteq V$ of query vertices, and a positive integer $h \in \mathbb{N}^+$, let $\{\langle S_i, \Delta_i\rangle\}_{i=1}^h$ be a solution to Problem 4.3 on input $\langle G, Q, h\rangle$ with S_i corresponding to the (Q, Δ_i)-highest-order-span-core of G, $\forall i \in [1, h]$. For every other solution $\{\langle S'_i, \Delta_i\rangle\}_{i=1}^h$ (referring to the same segmentation $\{\Delta_i\}_{i=1}^h$) to Problem 4.3 on input $\langle G, Q, h\rangle$ it holds that $S'_i \subseteq S_i$, $\forall i \in [1, h]$.*

Proof. Let k_i be the minimum degree of S_i, i.e., $k_i = v^*_{Q,\Delta_i}$ is the order of the (Q, Δ_i)-highest-order-span-core. Assume that there exists a solution S'_i to Problem 4.4 that is not contained in S_i. This implies that (i) the minimum degree of a vertex of S'_i in Δ_i is k_i, and (ii) the minimum degree of a vertex of $S_i \cup S'_i$ in Δ_i is k_i as well. This violates the maximality condition of the definition of span-core, since, by hypothesis, S_i corresponds to the (Q, Δ_i)-highest-order-span-core of G. □

The above proposition states that, given a solution $\{\langle S_i, \Delta_i\rangle\}_{i=1}^h$ to the Temporal Community Search problem where every S_i corresponds to the (Q, Δ_i)-highest-order-span-core of the input graph, one can focus on the various S_i solely to refine the output communities, as such S_i are guaranteed to contain *all* optimal solutions of the underlying problem instance (while keeping the segmentation $\{\Delta_i\}_{i=1}^h$ fixed). Within this view, we formulate the following optimization problem (which is a variant of Problem 4.4, with the additional constraint of requiring a smallest-sized solution):

Algorithm 4.5: Greedy-minimum-community-search

```
Input: A temporal graph $G = (V, E, T)$, a set $Q \subseteq V$ of query vertices, an interval
       $\Delta \sqsubseteq T$, a subset of vertices $S^* \subseteq V$ containing all the solutions to Problem 4.4 on
       input $\langle G, Q, \Delta \rangle$.
Output: A subset $S^*_{min}$ of vertices such that $Q \subseteq S^*_{min} \subseteq S^*$ and
        $\min_{u \in S^*_{min}} d_\Delta(S^*_{min}, u) \geq \min_{u \in S^*} d_\Delta(S^*, u)$.
1  $S^*_{min} \leftarrow \emptyset$;   $P \leftarrow \emptyset$;   $\mathcal{A} \leftarrow \emptyset$
2  add every $q \in Q$ to $P$ with priority $+\infty$
3  $k^* \leftarrow \min_{u \in S^*} d_\Delta(S^*, u)$;   $k^*_{min} \leftarrow 0$
4  while $k^*_{min} < k^*$ or $Q \not\subseteq S^*_{min}$ do
5      dequeue $u$ from $P$
6      $S^*_{min} \leftarrow S^*_{min} \cup \{u\}$
7      forall $v \in neigh_\Delta(S^*, u) \setminus S^*_{min} \setminus P$ do
8          $\mathcal{A}[v] \leftarrow score(v)$
9          add $v$ to $P$ with priority $\mathcal{A}[v]$
10     forall $v \in neigh_\Delta(S^*_{min}, u)$ do
11         if $d_\Delta(S^*_{min}, v) = k^*$ then
12             forall $w \in neigh_\Delta(P, v)$ do
13                 $\mathcal{A}[w] \leftarrow \mathcal{A}[w] - 1$
14     $k^*_{min} \leftarrow \min_{v \in S^*_{min}} d_\Delta(S^*_{min}, v)$
```

Problem 4.7. *Given a temporal graph $G = (V, T, \tau)$, a set $Q \subseteq V$ of query vertices, and an interval $\Delta \sqsubseteq T$, let $S^* \subseteq V$ be the subset of vertices containing all the solutions to Problem 4.4 on input $\langle G, Q, \Delta \rangle$ (according to what stated in Proposition 4.2). Find*

$$S^*_{min} = \text{argmin}_{\{S | Q \subseteq S \subseteq S^*, \min_{u \in S} d_\Delta(S,u) \geq \min_{u \in S^*} d_\Delta(S^*,u)\}} |S|. \tag{4.7}$$

Theorem 4.3. *Problem 4.7 in* **NP***-hard.*

Proof. Consider (the optimization version of) the **NP**-hard mCST problem introduced by Cui *et al.* [64]: given a graph $H = (V_H, E_H)$ and a query vertex $q \in V_H$, find a minimum-sized subgraph that contains q, is connected, and maximizes the minimum degree. Given an instance $\langle H, q \rangle$ of the mCST problem, construct an instance $\langle G, Q, \Delta \rangle$ of Problem 4.7 by defining G as composed by a single temporal snapshot corresponding to graph H, Δ as a singleton interval composed of the single timestamp of G, and setting $Q = \{q\}$. It is straightforward to see that solving Problem 4.7 on input $\langle G, Q, \Delta \rangle$ is equivalent to solving mCST on input $\langle H, q \rangle$, as the constraint about connectedness is automatically satisfied in Problem 4.7 for the special case of a single query vertex. □

As Problem 4.7 is **NP**-hard, we devise a heuristic that is inspired to the greedy one proposed for the Minimum Community Search problem in [25]. The proposed heuristic is outlined in Algorithm 4.5 and described next. In the pseudocode and in the following we denote as k^* and k^*_{min} the minimum degree of S^* and S^*_{min}, respectively, and as $neigh_\Delta(S, u)$ the neighbors of a vertex $u \in V$ in the subgraph induced by $S \subseteq V$ and $\Delta \sqsubseteq T$. Algorithm 4.5 iteratively adds vertices to the solution S^*_{min} according to a priority queue P. Priorities of vertices in P are defined based on a score that measures how promising a vertex is for making the current solution S^*_{min} reach the optimal minimum degree. Specifically, the score of a vertex $u \in S^*$ is defined as:

$$score(u) = score^+(u) - score^-(u), \tag{4.8}$$

where

$$score^+(u) = |\{v \in neigh_\Delta(S^*_{min}, u) \mid d_\Delta(S^*_{min}, v) < k^*\}|; \tag{4.9}$$

$$score^-(u) = \max\{0, k^* - d_\Delta(S^*_{min}, u)\}. \tag{4.10}$$

Table 4.1: Temporal graphs used in the experiments.

dataset	$\|V\|$	$\|E\|$	$\|T\|$	window size	domain
HighSchool	327	47k	1212	5 mins	face-to-face
PrimarySchool	242	55k	390	5 mins	face-to-face
HongKong	806	2M	2976	5 mins	face-to-face
ProsperLoans	89k	3M	307	7 days	economic
Last.fm	992	4M	77	21 days	co-listening
WikiTalk	2M	10M	192	28 days	communication
DBLP	1M	11M	80	366 days	co-authorship
StackOverflow	2M	16M	51	56 days	question answering
Wikipedia	343k	18M	101	56 days	co-editing
Amazon	2M	22M	115	28 days	co-rating
Epinions	120k	33M	25	21 days	co-rating

$score^+(u)$ is the gain effect of adding u to S^*_{min}, while $score^-(u)$ is the penalty effect. In particular, $score^+(u)$ counts the number of neighbors of u in S^*_{min} that would benefit from the inclusion of u to S^*_{min}, i.e., that have degree less than k^*. On the other hand, $score^-(u)$ represents the number of neighbors of u still required in S^*_{min} so that u has degree at least k^*. The algorithm starts by adding the query vertices to the queue P with priority $+\infty$, in order to ensure that they will be selected at the very beginning. At each iteration of the main cycle of the algorithm (starting at Line 4), the vertex u exhibiting the highest priority is dequeued from P and is added to the solution S^*_{min}. As a consequence, a couple of updates are performed. First, u's neighbors not in the priority queue P are added to it (Lines 8-9). Note that this is the only step of the algorithm where the score of a vertex is computed from scratch and stored in $\mathcal{A}$, a map that keeps the scores of all vertices in P up-to-date during the whole execution of the algorithm. The second update consists in recomputing the score of every v's neighbor w in the queue, if a vertex $v \in S^*_{min}$ has reached the desired minimum degree k^* after the addition of u.

4.5 Experiments

In this section we present an experimental evaluation to empirically assess the performance of all the proposed methods. Specifically, we focus on whole span-core decomposition (Section 4.5.1), maximal span-cores (Section 4.5.2), characterization of the extracted span-cores (Section 4.5.3), and temporal community search (Section 4.5.4).

Datasets. We use eleven real-world datasets recording timestamped interactions between entities. For each dataset we select a window size to define a discrete time domain, composed of contiguous timestamps of the same duration, and build the corresponding temporal graph. If multiple interactions occur between two entities during the same discrete timestamp, they are counted as one. The characteristics of the resulting temporal graphs, along with the selected window sizes, are reported in Table 4.1.

The three smallest datasets were gathered by using wearable proximity sensors in schools, with a temporal resolution of 20 seconds. PrimarySchool[1] contains the contact events between 242 volunteers (232 children and 10 teachers) in a primary school in Lyon, France, during two days [217]. HighSchool[1] describes the close-range proximity interactions between students and teachers (327 individuals overall) of nine classes during five days in a high school in Marseilles, France [174]. HongKong reports the same kind of interactions for a primary school in Hong Kong, whose population consists of 709 children and 65 teachers divided into thirty classes, for eleven consecutive days [204]. ProsperLoans[2] represents the network of loans between the users of Prosper, a marketplace of loans between privates. Last.fm[2] records the co-listening activity of the Last.fm streaming platform: an edge exists between two users if they listened to songs of the same band within the same discrete timestamp. WikiTalk[2] is the communication network of the English

[1]sociopatterns.org
[2]konect.cc

Wikipedia. DBLP[2] is the co-authorship network of the authors of scientific papers from the DBLP computer science bibliography. StackOverflow[3] includes the answer-to-question interactions on the stack exchange of the stackoverflow.com website. Wikipedia[2] connects users of the Italian Wikipedia that co-edited a page during the same discrete timestamp. Finally, for both Amazon[2] and Epinions[2], vertices are users and edges represent the rating of at least one common item within the same discrete timestamp.

Implementation. All methods are implemented in Python (v. 2.7.16) and compiled by Cython. All the experiments were run on a machine equipped with Intel Xeon CPU at 2.1GHz. The experiments reported in Sections 4.5.1 and 4.5.2 used 64GB RAM, while the ones in Section 4.5.4 used 32GB RAM.

Reproducibility. Our code is available at github.com/egalimberti/span_cores.

4.5.1 Span-core decomposition

We compare the two methods to compute a complete decomposition described in Section 4.2, i.e., the baseline Naïve-span-cores and the proposed Span-cores, in terms of execution time, memory, and total number of vertices input to the core-decomposition subroutine. We report these measures, together with the number of span-cores and maximal span-cores of each dataset, in Table 4.2.

In terms of execution time, Span-cores considerably outperforms Naïve-span-cores in all datasets, achieving a speed-up from 2.1 up to two orders of magnitude. The speed-up is explained by the number of vertices processed by the core-decomposition subroutine, which is the most time-consuming step of the algorithms albeit linear in the size of the input subgraph. The difference of this quantity between Span-cores and Naïve-span-cores reaches over an order of magnitude in the WikiTalk, Wikipedia, and Epinions dataset, confirming the effectiveness of the “horizontal containment” relationships. The memory required by the two procedures is comparable in all cases since the largest structures needed in memory are the temporal graph itself and the set $\mathbf{C}$ of all span-cores.

4.5.2 Maximal span-cores

We compare our Maximal-span-cores algorithm to the naïve approach, described at the beginning of Section 4.3, based on running the Span-cores algorithm and filtering out the non-maximal span-cores, which we refer to as Naïve-maximal-span-cores. The results are again reported in Table 4.2.

Naïve-maximal-span-cores behaves very similarly to Span-cores: they only differ for the filtering mechanism which requires a few additional seconds in most cases. Maximal-span-cores is much faster than Naïve-maximal-span-cores for all datasets, with a speed-up from 1.3 for the Epinions dataset to one order of magnitude for the HongKong dataset. Except for the school datasets and Last.fm, the difference in terms of number of processed vertices is between one and three orders of magnitude, attesting the advantages of the top-down strategy of Maximal-span-cores, which avoids the visit of portions of the span-core search space and handles the overhead of reconstructing graphs, i.e., $(V_{lb}, E_\Delta[V_{lb}])$, efficiently. Finally, the memory requirements of the two methods are comparable for all datasets.

4.5.3 Span-cores characterization

We compare and characterize all span-cores against maximal span-cores. At first, Table 4.2 shows that span-cores are at least one order of magnitude more numerous than maximal span-cores for all datasets, with the maximum difference of three orders of magnitude for the HongKong dataset.

In Figure 4.2 we show the number (top) and the average size (bottom) of span-cores and maximal span-cores as a function of the order k for the DBLP and Epinions datasets. For both datasets, the number of maximal span-cores is at least one order of magnitude lower than the total number of span-cores up to a quarter of the k domain, where the span-cores are more numerous. Instead, in the rest of the domain, they mostly coincide due to the maximality condition over $|\Delta|$. The average size is also smaller for maximal span-cores, difference that wears thin when the gap

[3]snap.stanford.edu

Table 4.2: Evaluation of the proposed algorithms: number of output span-cores, running time, memory, and number of processed vertices.

dataset	method	# output span-cores	running time (s)	memory (GB)	# processed vertices
HighSchool	Naïve-span-cores	12 320	18	0.1	3M
	Span-cores		1	0.1	581k
	Naïve-maximal-span-cores	450	1	0.1	581k
	Maximal-span-cores		0.3	0.1	181k
PrimarySchool	Naïve-span-cores	4 703	4	0.1	818k
	Span-cores		0.6	0.1	174k
	Naïve-maximal-span-cores	409	0.6	0.1	174k
	Maximal-span-cores		0.1	0.1	63k
HongKong	Naïve-span-cores	2 367 743	85 180	1	819M
	Span-cores		18 389	0.8	216M
	Naïve-maximal-span-cores	1 807	18 641	0.8	216M
	Maximal-span-cores		339	0.5	212M
ProsperLoans	Naïve-span-cores	4 273	101	2	55M
	Span-cores		46	2	27M
	Naïve-maximal-span-cores	293	48	2	27M
	Maximal-span-cores		8	2	980k
Last.fm	Naïve-span-cores	126 819	707	0.5	2M
	Span-cores		199	0.5	531k
	Naïve-maximal-span-cores	1 670	202	0.5	531k
	Maximal-span-cores		57	0.5	271k
WikiTalk	Naïve-span-cores	19 693	322 302	36	25B
	Span-cores		1 084	36	555M
	Naïve-maximal-span-cores	632	1 194	36	555M
	Maximal-span-cores		126	35	2M
DBLP	Naïve-span-cores	6 135	10 506	11	1B
	Span-cores		278	11	150M
	Naïve-maximal-span-cores	268	292	11	150M
	Maximal-span-cores		116	11	620k
StackOverflow	Naïve-span-cores	1 238	5 360	10	1B
	Span-cores		245	10	127M
	Naïve-maximal-span-cores	129	245	10	127M
	Maximal-span-cores		128	10	3M
Wikipedia	Naïve-span-cores	125 191	17 155	4	1B
	Span-cores		522	4	35M
	Naïve-maximal-span-cores	2 147	537	4	35M
	Maximal-span-cores		201	4	320k
Amazon	Naïve-span-cores	29 318	10 415	18	2B
	Span-cores		409	18	247M
	Naïve-maximal-span-cores	303	580	18	247M
	Maximal-span-cores		123	18	688k
Epinions	Naïve-span-cores	63 111	699	4	39M
	Span-cores		186	4	3M
	Naïve-maximal-span-cores	320	201	4	3M
	Maximal-span-cores		154	5	129k

between the numbers of span-cores and maximal span-cores starts decreasing since, for high values of k, most (or all) span-cores are maximal.

Figure 4.3 shows a different picture when numbers and average sizes of span-cores are shown as a function of the size of the span $|\Delta|$. For both datasets, the number of span-cores and maximal span-cores is decreasing – which is expected since the number of intervals decreases when $|\Delta|$ increases – with a constant gap close to one and two orders of magnitude, respectively. On the

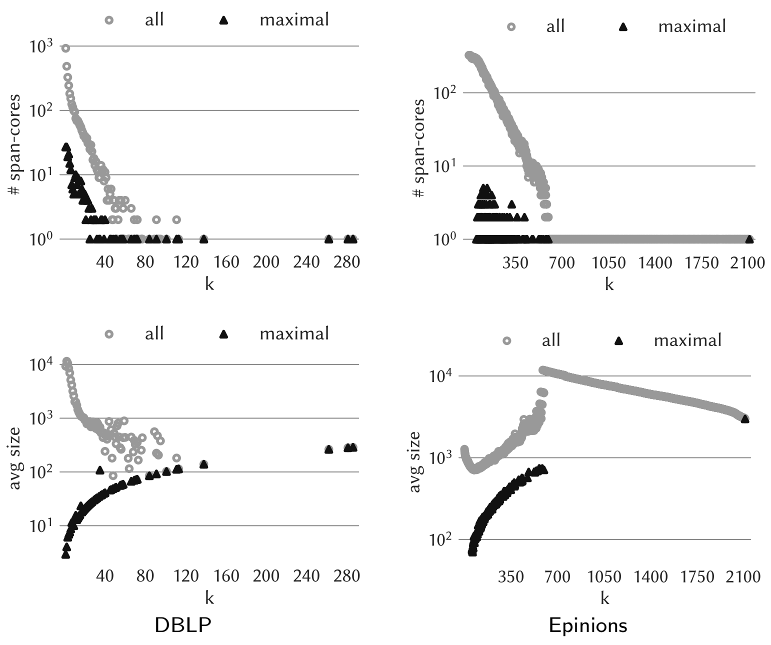


Figure 4.2: Top plots: number of span-cores and maximal span-cores as a function of the order k. Bottom plots: average size of all span-cores and maximal span-cores as a function of the order k.

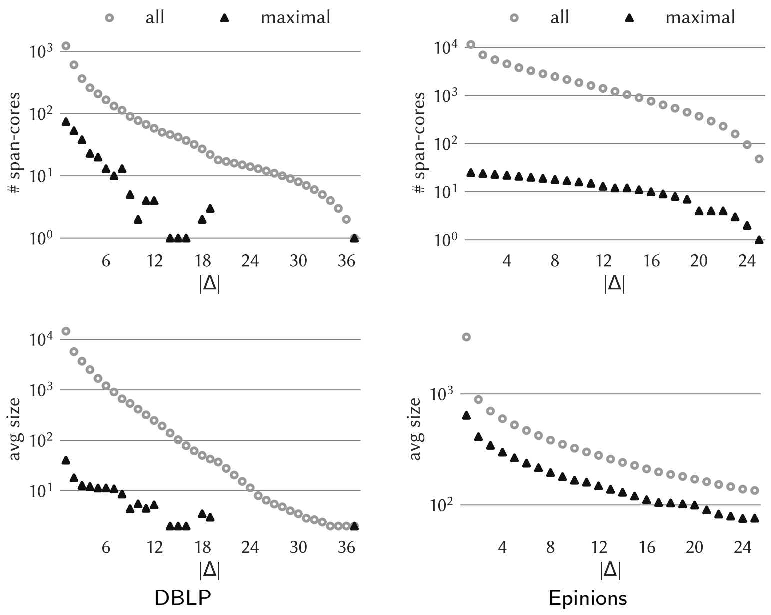


Figure 4.3: Top plots: number of span-cores and maximal span-cores as a function of the size of the temporal span $|\Delta|$. Bottom plots: average size of all span-cores and maximal span-cores as a function of the size of the temporal span $|\Delta|$.

other hand, the behavior of the average size is quite different between the two datasets. For low values of $|\Delta|$, the average size of span-cores of the DBLP dataset is much higher than the average size of maximal span-cores, then the difference decreases and vanishes at the end of domain where a maximal span-core of $|\Delta| = 37$ dominates all other span-cores with $|\Delta| \geq 20$. Instead, for the Epinions dataset, the average size of all span-cores and of maximal span-cores follow the same behavior, with a difference of less than an order of magnitude, because the maximality condition over k excludes the largest span-cores from the set of maximal span-cores.

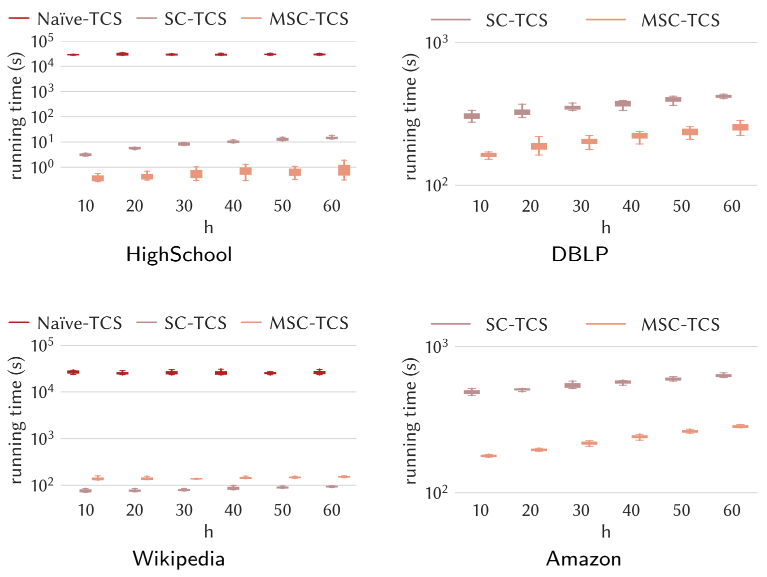


Figure 4.4: Running time of the algorithms for Temporal Community Search, as a function of the number h of output communities. Each boxplot corresponds to 15 data points.

4.5.4 Temporal community search

In this subsection we assess the performance of the proposed algorithms for temporal community search (presented in Sections 4.4.2–4.4.3), as well as the greedy procedure for reducing the size of the output communities (presented in Section 4.4.4). In the remainder of this subsection we refer to our basic algorithm (i.e., Algorithm 4.3, which precomputes the penalty scores via span-core decomposition) as SC-TCS, and to our more efficient algorithm (i.e., Algorithm 4.4, which exploits maximal span-cores to reduce the number of timestamps to be considered) as MSC-TCS. We also involve in the comparison a naïve version of Algorithm 4.3, where the penalty scores of the various intervals are computed from scratch during the execution of the algorithm, instead of precomputing them all via span-core decomposition. We refer to such a naïve method as Naïve-TCS.

The experimental setting we consider here is as follows. We vary the number $|Q|$ of query vertices from 1 to 3. In particular, when $|Q| = 1$, we sample the single query vertex uniformly at random from the whole vertex set V. Instead, for $|Q| > 1$, we employ a more sophisticated sampling strategy that aims at finding meaningful query-vertex sets, i.e., vertices interacting with each other during the temporal observations, and, at the same time, independent from the specific form of the resulting span-core decomposition. Specifically, the sampling strategy we use is based on an adaptation of random walk to the temporal settings:

- Select a vertex uniformly at random from the whole V and add such a vertex to the set Q_{visited} of visited vertices
- Starting from the first timestamp of the temporal domain T, iteratively:
  - With probability p, move the random walker to a neighbor of the current vertex and add the neighbor to Q_{visited}. If the current vertex has no neighbors in a given timestamp, the random walker jumps to the first next timestamp in which that vertex has at least one neighbor
  - With probability $1 - p$, keep the random walker at the current vertex, but go to the next timestamp
- Restart if the last timestamp of T is reached
- Stop when $|Q_{\text{visited}}|$ reaches a proper (user-defined) size ν

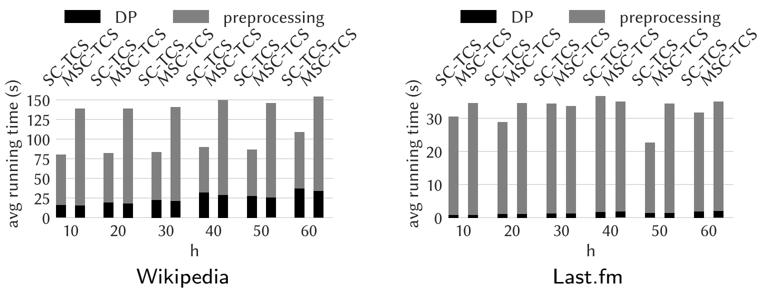


Figure 4.5: Split of the average running time of the SC-TCS and MSC-TCS algorithms into dynamic programming (DP) and precomputation, for the Wikipedia and Last.fm datasets.

- Sample $|Q|$ query vertices from Q_{visited} with probability proportional to the frequency of the visits during the random walk

In our experiments we set $p = 0.8$ and $\nu = 3|Q|$. As far as the number h of output communities, we consider the range $h \in [10, 20, 30, 40, 50, 60]$ on all datasets, with the exception of StackOverflow, for which we discard $h = 60$, and Epinions, for which we consider $h \in [4, 8, 12, 16, 20, 24]$. For every parameter configuration, we perform five runs of every algorithm (in every run we sample a different query-vertex set). Note that we were not able to run the algorithms for temporal community search on the WikiTalk dataset due to memory constraints.

Running time. In Figure 4.4 we show the running time of the proposed algorithms as a function of the number h of output communities, for the HighSchool, DBLP, Wikipedia, and Amazon datasets. The first general observation we make is that the running time of all algorithms increases as h gets higher. This in accordance with the time-complexity analysis reported in Section 4.4. Also, running times are independent of the selected query-vertex set Q. Looking at the individual performance, we notice that, as expected, the Naïve-TCS method has severe limitations in terms of efficiency: it takes hours to run on the HighSchool and Wikipedia datasets, while it is not able to terminate in less than 10 days on the remaining datasets. SC-TCS and MSC-TCS are much faster than Naïve-TCS, achieving a speedup of up to more than four orders of magnitude. MSC-TCS is in most cases faster than SC-TCS, with speedup up to one order of magnitude (on HighSchool, for $h = 60$). This confirms that the exploitation of the maximal span-cores is effective in both shortening the precomputation time and reducing the temporal domain considered in the dynamic-programming step. The only exception is the Wikipedia dataset. To dive deeper into the motivations of this exception, we report in Figure 4.5 the split of the average running time of SC-TCS and MSC-TCS into the time spent in the dynamic-programming step (DP) (which also includes the identification of the reduced temporal domain T^* for MSC-TCS), and the precomputation time (i.e., the time required for computing all penalty scores via span-core decomposition or maximal span-cores). Interestingly, what affects the most the running time is the precomputation of the scores. Apparently, the Q-constrained version of Span-cores is more efficient than Maximal-span-cores in some datasets, which we believe is due to the structure of the search space. On the other hand, these results confirm that the reduction of the temporal domain considered by the dynamic-programming step is actually effective since the DP running time of MSC-TCS is always less than (or equal to) the DP running time of SC-TCS.

Greedy-minimum-community-search. Here we evaluate the performance of the proposed Greedy-minimum-community-search algorithm (Algorithm 4.5) for reducing the size of the output communities. We recall that the proposed algorithms for Temporal Community Search (evaluated above) output communities corresponding to the (Q, Δ_i)-highest-order-span-cores for all $\{\Delta_i\}_{i=1}^{h}$ temporal intervals identified. The Greedy-minimum-community-search algorithm takes every (Q, Δ_i)-highest-order-span-core and attempts to reduce its size, while preserving optimality. Thus, the ultimate goal of the evaluation presented next is to show how well Greedy-minimum-community-search is able to reduce the size of the original span-cores, and what is its overhead in terms of running time.

Figure 4.6 compares the size of the starting (Q, Δ_i)-highest-order-span-cores and the size of the

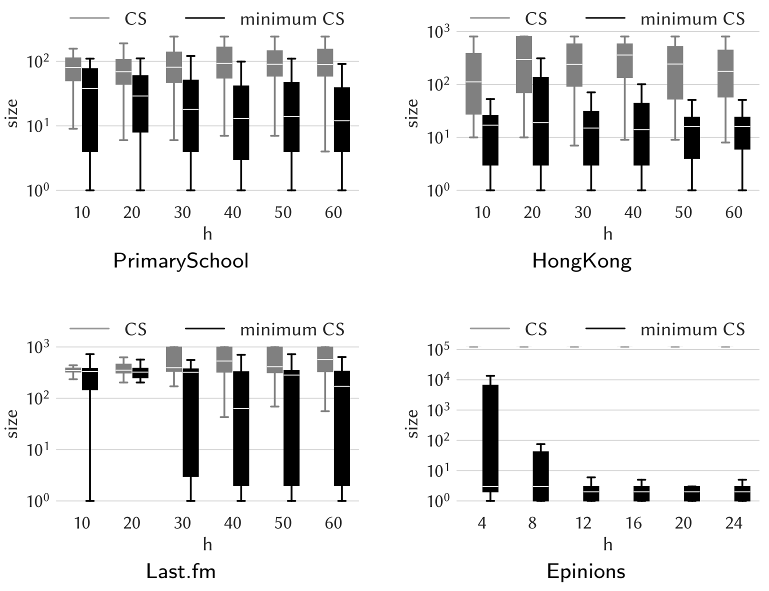


Figure 4.6: Comparison of the size of the communities in the solutions to Temporal Community Search: original output of the algorithms for Temporal Community Search (CS) and after running the Greedy-minimum-community-search algorithm on top of them (minimum CS). Each boxplot corresponds to 15 data points.

Table 4.3: Average running time of an execution of the Greedy-minimum-community-search algorithm.

	HighSchool	PrimarySchool	HongKong	ProsperLoans	Last.fm
running time (s)	0.003	0.001	0.02	0.3	0.06

	DBLP	StackOverflow	Wikipedia	Amazon	Epinions
running time (s)	7	8	1	7	6

corresponding reduced community yielded by the Greedy-minimum-community-search algorithm, for the PrimarySchool, HongKong, Last.fm, and Epinions datasets. It can be easily observed that, as a general trend, the reduced communities are much smaller than the original ones, in all datasets, up to four orders of magnitude. The results on the Epinions dataset are a bit different than the other three datasets. In fact, on that dataset, the original communities (CS) always include the whole 120k vertices of the graph, while the communities found by Greedy-minimum-community-search (minimum CS) have median size smaller than 10, and, in many cases, they correspond to communities composed of the query vertices only. This means that, on the Epinions dataset, for our tested queries, the algorithms for Temporal Community Search do not extract communities that are really cohesive around the query vertices. This way, the benefits of exploiting an a-posteriori community-size-reduction step are less evident. Also, we do not notice any evident pattern as a function of h, for any dataset.

In Table 4.3 we report the average running time of an execution of Greedy-minimum-community-search, for all datasets. Note that this is the average time required to process one of the h communities in a solution to Temporal Community Search. Greedy-minimum-community-search runs in 8 seconds or less in all tested datasets. Therefore, the additional running time required by the algorithm is rather negligible.

To summarize, Greedy-minimum-community-search is empirically recognized as a powerful post-processing method for improving the quality of the solutions to Temporal Community Search: it finds much smaller communities at a very small additional computational cost.

4.6 Summary

Temporal networks are a powerful representation of how relations are established and interrupted along time among a given population of entities. An interesting primitive for analyzing this type of networks is the extraction of relevant patterns, such as dense subgraphs, together with their time interval of existence (or span). Following this idea, we introduced a notion of temporal core decomposition where each core is associated with its span. Exploiting containment properties among cores we developed efficient algorithms for computing all the span-cores, and also only the maximal ones. We then introduced the problem of temporal community search and showed how it can be solved in polynomial time via dynamic programming. We also proved an interesting connection between temporal community search and maximal span-cores, which made it possible to devise a considerably more efficient algorithm than the naïve dynamic-programming one.

In Chapter 7 we show the usefulness of the definitions proposed here by direct application to practical problems in face-to-face interaction networks, i.e., the HighSchool, PrimarySchool, and HongKong datasets introduced in Section 4.5.

Chapter 5

Polarized communities in signed networks

The increase of polarization around controversial issues is a growing concern with important societal fallouts. While controversy can be engaging, and can lead to users spending more time on social-media platforms, in disproportionate amounts it can generate a negative user experience, potentially leading to the abandonment of the platform. Excessive polarization, together with the emergence of bots and the spread of misinformation, has thus become an urgent technological problem that needs to be solved. It is not surprising that the last few years have witnessed an uptake of the research on methods for the detection and suppression of these phenomena [103, 163, 164, 233, 182, 116]. While polarization is a well studied phenomenon in political and social sciences [23, 50, 80, 88, 104, 236], modern social-media platforms brought it to a different scale, providing an unprecedented wealth of data. The necessity to analyze the available data and gain valuable insights brings new algorithmic challenges.

In order to study polarization in large-scale online data, one first step is to detect it. In this chapter we study a fundamental problem abstraction for this task, i.e., the problem of *discovering polarized communities in signed networks*. A signed network is a simple, yet general and powerful, representation: vertices represent entities and edges between vertices represent interactions, which can be friendly (positive) or antagonistic (negative) [121]. Signed graphs analysis has many applications from modeling interactions in social media [151], to mining user reviews [32], to studying information diffusion and epidemics [162], to recommending products in e-commerce sites [169, 232], and to estimating the structural balance of a (physical) complex system [13, 172].

In this chapter, we introduce the 2-Polarized-Communities problem (2PC), which requires finding two communities (subsets of the network vertices) such that within communities there are mostly positive edges while across communities there are mostly negative edges. Furthermore, we do not aim to partition the whole network, so the two polarized communities we are searching can be concealed within a large body of other network vertices, which are neutral with respect to the polarized structure. Our hypothesis is that such 2-community polarized structure accurately captures controversial discussions in real-world social-media platforms.

Figure 5.1 shows an example of the two most polarized communities found in the Congress network (details in Section 5.4). The two communities involve 34 and 37 vertices (out of 219), respectively, having more than 98% of positive edges within and 78% negative edges across. The vertices in gray do not participate in any of the two polarized communities: either they have too few connections with any community, or the polarity of their relations are mixed and thus their position within the debate unclear.

This work is, to the best of our knowledge, the first to propose a spectral method for extracting polarized communities from signed networks. In addition, we present hardness results and approximation guarantees. Our problem formulation deviates from the bulk of the literature where methods typically look for finding many communities while partitioning the whole network [9, 24, 58, 60, 112, 152]. The closest approach to our problem statement is the work by Coleman et al. [60], who employ the correlation-clustering framework and search for exactly two communities. However, while in that work all vertices must be included in a cluster, in our setting

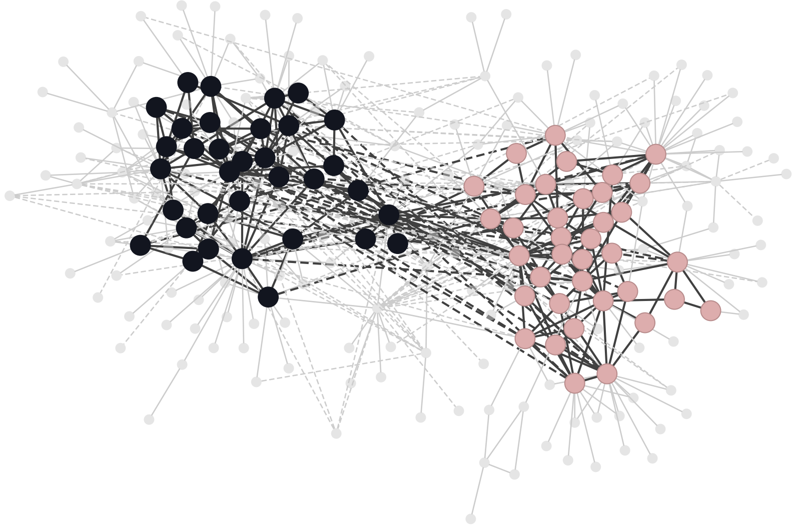


Figure 5.1: An example of two polarized communities in the Congress network (dataset details in Section 5.4). Solid edges are positive, while dashed edges are negative.

we allow vertices not to be part of any cluster. This captures the fact that polarized communities are typically concealed within a large body of neutral vertices in a social network. An algorithm that attempts to partition the whole network would fail to reveal these communities. As an additional feature, our methods can be fine-tuned to increase or decrease the size of the discovered communities. Finally, while some spectral techniques promote balanced partitions, we hypothesize that two polarized communities might be of very different sizes, and thus our problem formulation does not enforce evenly sized subgraphs.

Our reliance on spectral methods carries several benefits. First, it is possible to leverage readily available, highly optimized, and parallelized software implementations. This makes it straightforward for the practitioner to analyze large networks in real settings using our approach. Second, even though by this work we focus on the case of two communities, we can take inspiration from the existing literature on spectral graph partitioning to easily extend our algorithms to the case of an arbitrary number of subgraphs, e.g., by recursive two-way partitioning or the analysis of multiple eigenvectors [214].

In this chapter we make the following contributions:

- We formulate the 2-Polarized-Communities problem (2PC) as a “discrete eigenvector” problem (Section 5.1).
- Exploiting a reduction from classic correlation clustering, we prove that 2PC is **NP**-hard (Theorem 5.1).
- We devise two intuitive spectral algorithms (Section 5.2), one deterministic, and one randomized with quality guarantee $\sqrt{n}$ (Theorem 5.2), which is tight up to constant factors. We believe these to be the first purely combinatorial bounds for spectral methods. Our results apply to graphs of arbitrary weights. Our algorithms' running time is essentially the time required to compute the first eigenvector of the adjacency matrix of the input graph.
- Our experiments (Section 5.4) on a large collection of real-world signed networks show that the proposed algorithms discover higher quality solutions, are much faster than the baselines, and can scale to much larger networks. In addition, they are able to identify ground-truth planted polarized communities in synthetic datasets.

5.1 2-Polarized-Communities

Our setting is reminiscent to the *correlation-clustering* problem [24], which we recall here. Given a signed network $G = (V, E_+, E_-)$, where E_+ is the set of positive edges and E_- the set of negative edges, the goal is to find a partition of the vertices into k clusters, so as to maximize the number of positive edges within clusters plus the number of negative edges between clusters.

An interesting property of the correlation-clustering formulation is that one does not need to specify in advance the number of clusters k, instead it is part of the optimization. In certain cases, however, the number of clusters is given as input. The general problem (given k) has been studied by Giotis and Guruswami [112], while Coleman et al. [60] studied the 2-Correlation-Clustering problem ($k = 2$). The problem arises, for instance, in the domain of social networks, where two well-separated clusters reveal a polarized structure. It can be defined as follows.

Problem 5.1 (2CC)**.** *Given a signed network $G = (V, E_+, E_-)$, find a partition S_1, S_2 of V so as to maximize*

$$cc(S_1, S_2) = \sum_{\substack{i\in\{1,2\}\\(u,v)\in S_i\times S_i}} \frac{1}{2}\mathbf{1}_{E_+}(u,v) + \sum_{(u,v)\in S_1\times S_2} \mathbf{1}_{E_-}(u,v), \tag{5.1}$$

where $\mathbf{1}_S$ is the indicator function of the set S.

A crucial limitation of the 2CC problem is that all vertices must be accounted for in one of the two clusters. From an application perspective, however, this may be a strong assumption. For example, in a social network, we may expect two polarized communities on a topic, but there may be many individuals who are neutral.

In order to find communities embedded within large networks, we need to exclude neutral vertices from the solution. Therefore, a first approach might be to consider maximizing agreements including a neutral cluster, that is, finding a partition of V into S_1, S_2, and S_0, so that S_1 and S_2 are the two polarized communities, and S_0 is the neutral community, and the 2CC objective $cc(S_1, S_2)$ is maximized. However, this modification does not change the problem significantly: it is always no worse to switch a vertex from cluster S_0 to one of the other two clusters.

Proposition 5.1. *Let S_0, S_1, S_2 be any partition of V, with $S_0 \neq \emptyset$. Then there is always a partition S_1', S_2' of V (i.e., $S_1' \cup S_2' = V$ and $S_1' \cap S_2' = \emptyset$) with $S_1 \subseteq S_1'$ and $S_2 \subseteq S_2'$ so that*

$$cc(S_1', S_2') \geq cc(S_1, S_2). \tag{5.2}$$

A further modification might be to subtract disagreements from the value of the solution, that is, to maximize agreements minus disagreements. In other words, we consider the following problem.

Problem 5.2 (2CC-Full)**.** *Given a signed network $G = (V, E_+, E_-)$, find a partition S_0, S_1, S_2 of V so as to maximize*

$$\begin{aligned}\overline{cc}(S_1, S_2) \quad = \quad & \sum_{\substack{i\in\{1,2\}\\(u,v)\in S_i\times S_i}} \frac{1}{2}\left(\mathbf{1}_{E_+}(u,v) - \mathbf{1}_{E_-}(u,v)\right) \\ & + \sum_{(u,v)\in S_1\times S_2} \left(\mathbf{1}_{E_-}(u,v) - \mathbf{1}_{E_+}(u,v)\right),\end{aligned} \tag{5.3}$$

where $\mathbf{1}_S$ is the indicator function of the set S.

Unfortunately, problem 2CC-Full suffers from the same issue as problem 2CC: switching a vertex from the neutral cluster S_0 to one of the polarized clusters S_1 or S_2 (the one that is best) leads to no worse solution according to the objective $\overline{cc}$.

Proposition 5.2. *Let S_0, S_1, S_2 be any partition of V, with $S_0 \neq \emptyset$. Then there is always a partition S_1', S_2' of V (i.e., $S_1' \cup S_2' = V$ and $S_1' \cap S_2' = \emptyset$) with $S_1 \subseteq S_1'$ and $S_2 \subseteq S_2'$ so that*

$$\overline{cc}(S_1', S_2') \geq \overline{cc}(S_1, S_2). \tag{5.4}$$

Algorithm 5.1: EIGENSIGN

```
Input: An adjacency matrix $A$.
Output: An $n$-dimensional vector $\vec{x}$.
1 compute $\vec{v}$, the eigenvector corresponding to the largest eigenvalue $\lambda_1$ of $A$
2 construct $\vec{x}$ as follows: for each $i \in \{1, \ldots, n\}$, $x_i = sgn(v_i)$
```

A nice property of the $\overline{cc}$ objective is that it can be written neatly in a matrix notation. Let A be the adjacency matrix of the signed network $G = (V, E_+, E_-)$, where positive edges $(i, j) \in E_+$ are indicated by $A_{ij} = 1$, negative edges $(i, j) \in E_-$ are indicated by $A_{ij} = -1$, and non-edges are indicated by $A_{ij} = 0$. A partition S_0, S_1, S_2 of V can be represented by a vector $\vec{x} \in \{-1, 0, 1\}^n$, whose i-th coordinate is $x_i = 0$ if $i \in S_0$, $x_i = 1$ if $i \in S_1$, and $x_i = -1$ if $i \in S_2$. Then 2CC-FULL can be reformulated as follows.

Problem 5.3 (2CC-FULL)**.** *Given a signed network $G = (V, E_+, E_-)$ with n vertices and signed adjacency matrix A, find a partition S_0, S_1, S_2 of V represented by vector $\vec{x} \in \{-1, 0, 1\}^n$ maximizing*

$$\overline{cc}(S_1, S_2) = \vec{x}^T A\, \vec{x}. \tag{5.5}$$

Since our goal is to discover polarized communities S_1 and S_2 that are potentially concealed within other neutral vertices S_0, we want to find minimal sets S_1 and S_2. This can be achieved by normalizing $\vec{x}^T A\, \vec{x}$ with the size of S_1 and S_2, which in vector form is $\vec{x}^T \vec{x}$. This consideration motivates our last problem formulation, which we dub 2-POLARIZED-COMMUNITIES (2PC).

Problem 5.4 (2PC)**.** *Given a signed network $G = (V, E_+, E_-)$ with n vertices and signed adjacency matrix A, find a vector $\vec{x} \in \{-1, 0, 1\}^n$ that maximizes*

$$\frac{\vec{x}^T A\, \vec{x}}{\vec{x}^T \vec{x}}. \tag{5.6}$$

In the rest of this chapter we refer to the objective function of Problem 5.4 as *polarity.* As polarity is penalized with the size of the solution, vertices are only added to one of the two clusters if they contribute significantly to the objective. We show this problem to be **NP**-hard (proof in Section 5.3) and propose algorithms with approximation guarantees.

Theorem 5.1. 2PC *is* **NP**-*hard.*

It should be noted that 2PC does not enforce balance between the communities. This can be beneficial if there exist polarized communities of significantly different size in the input network. In an extreme case, the solution could even be comprised of a single cluster if there is a large, dense community that overwhelms any other polarized formation.

5.2 Algorithms for 2-POLARIZED-COMMUNITIES

The formulation of 2PC is suggestive of spectral theory, which we utilize to design our algorithms. We propose and analyze two spectral algorithms: one is deterministic, while the second is randomized and achieves approximation guarantee $\sqrt{n}$. The running time of both algorithms is dominated by the computation of a spectral decomposition of the adjacency matrix. In practice, this can be done using readily available implementations that exploit sparsity and can run in parallel on multiple cores.

The first algorithm, EIGENSIGN, works by simply discretizing the entries of the eigenvector of the adjacency matrix corresponding to the largest eigenvalue.

To illustrate the difficulty of approximating 2PC, we analyze the following simple algorithm, which we refer to as PICK-AN-EDGE. Pick an arbitrary edge: if it is positive, put the endpoints in one cluster, leaving the other cluster empty; if it is negative, put the endpoints in separate clusters.

Proposition 5.3. *The* PICK-AN-EDGE *algorithm gives an n-approximation of the optimum.*

Proof. The described algorithm outputs a solution $\vec{x}$ such that

$$\frac{\vec{x}^T A\,\vec{x}}{\vec{x}^T\vec{x}} \geq 1. \tag{5.7}$$

The result now follows from the fact that $OPT \leq \lambda_1 \leq n$, where λ_1 is the largest eigenvalue of A. □

In the case of networks with arbitrary real weights, it can be shown that despite the close relationship between the 2PC objective and the leading eigenvector of A, Eigensign cannot do better than this up to constant factors. Consider a fully connected network with one edge (u, v) of weight $w \gg 0$. The rest of the edges have weight close to zero. The primary eigenvector of the adjacency matrix has two entries — those corresponding to u and v — of the form $1/\sqrt{2} - \epsilon$ for some small ϵ, while the rest are close to zero. We construct a solution vector $\vec{y}$ as follows: the two entries corresponding to u and v are set to 1, and the rest to 0. We have $\vec{y}^T A\,\vec{y}/\vec{y}^T\vec{y} \approx w$. On the other hand, the Eigensign algorithm outputs a vector $\vec{x}$ for which $\vec{x}^T A\,\vec{x}/\vec{x}^T\vec{x} \approx 2w/n$. It should be noted however, that the focus of this work is the analysis of the 2PC problem on signed networks. The approximation capabilities of the Eigensign algorithm on signed networks (the adjacency matrix A contains entries with values only -1, 0, and 1) are left open.

Eigensign generally outputs a solution comprised of all the vertices in the graph — unless some components of the eigenvector are exactly zero — which is, of course, counter to the motivation of our problem setting.

To overcome this issue we propose a randomized algorithm, Random-Eigensign, which also computes the first eigenvector, i.e., $\vec{v}$, of the adjacency matrix. Instead of simply discretizing the entries of $\vec{v}$, it randomly sets each entry of $\vec{x}$ to 1 or -1 with probabilities determined by the entries of $\vec{v}$. Entries v_i with large magnitude $|v_i|$ are more likely to turn into $sgn(v_i)$ (-1 or 1), while entries v_i with small magnitude $|v_i|$ are more likely to turn into 0. For details see Algorithm 5.2. Note that if $\vec{x}$ is the output of Random-Eigensign, then $\mathbb{E}[\vec{x}] = \vec{v}$.

The next theorem shows approximation guarantees of Random-Eigensign for signed networks.

Theorem 5.2. *Algorithm* Random-Eigensign *gives a $\sqrt{n}$-approximation of the optimum in expectation.*

Proof. First, observe that we can rewrite the expected value of the objective as follows:

$$\begin{aligned}
\mathbb{E}\left[\frac{\vec{x}^T A\vec{x}}{\vec{x}^T\vec{x}}\right] &= \sum_{k=1}^{n} \mathbb{E}\left[\frac{\vec{x}^T A\vec{x}}{\vec{x}^T\vec{x}}\middle| \vec{x}^T\vec{x} = k\right] Pr(\vec{x}^T\vec{x} = k) && (5.8)\\
&= \sum_{k=1}^{n} \frac{1}{k}\mathbb{E}\left[\vec{x}^T A\vec{x} | \vec{x}^T\vec{x} = k\right] Pr(\vec{x}^T\vec{x} = k) && (5.9)\\
&= \sum_{k=1}^{n} \frac{1}{k}\sum_{i\neq j}\mathbb{E}\left[A_{ij}x_i x_j | \vec{x}^T\vec{x} = k\right] Pr(\vec{x}^T\vec{x} = k). && (5.10)
\end{aligned}$$

If we define $s_{ij} = sgn(v_i)sgn(v_j)$, where $sgn(x)$ denotes the sign of $x \in \mathbb{R}$, for all i, j we have

$$\begin{aligned}
&\mathbb{E}\left[A_{ij}x_i x_j | \vec{x}^T\vec{x} = k\right] Pr(\vec{x}^T\vec{x} = k)\\
&= A_{ij}s_{ij}Pr(x_i = 1, x_j = 1 | \vec{x}^T\vec{x} = k)Pr(\vec{x}^T\vec{x} = k). && (5.11)
\end{aligned}$$

Algorithm 5.2: RANDOM-EIGENSIGN

```
Input: An adjacency matrix $A$.
Output: An $n$-dimensional vector $\vec{x}$.
1 compute $\vec{v}$, the eigenvector corresponding to the largest eigenvalue $\lambda_1$ of $A$
2 construct $\vec{x}$ as follows: for each $i \in \{1, \dots, n\}$, run a Bernoulli experiment with success
    probability $|v_i|$. If it succeeds, then $x_i = sgn(v_i)$, otherwise $x_i = 0$
```

We now invoke Bayes' theorem and proceed.

$$\sum_{k=1}^{n} \frac{1}{k} \sum_{i \neq j} A_{ij} s_{ij} Pr(x_i = 1, x_j = 1) Pr(\vec{x}^T \vec{x} = k | x_i = 1, x_j = 1) \tag{5.12}$$

$$= \sum_{k=1}^{n} \frac{1}{k} \sum_{i \neq j} A_{ij} v_i v_j Pr(\vec{x}^T \vec{x} = k | x_i = 1, x_j = 1) \tag{5.13}$$

$$= \sum_{i \neq j} A_{ij} v_i v_j \sum_{k=1}^{n} \frac{1}{k} Pr(\vec{x}^T \vec{x} = k | x_i = 1, x_j = 1) \tag{5.14}$$

$$= \sum_{i \neq j} A_{ij} v_i v_j \mathbb{E} \left[\frac{1}{\vec{x}^T \vec{x}} | x_i = 1, x_j = 1 \right]. \tag{5.15}$$

Since $1/x$ is a convex function, by Jensen's inequality it is

$$\mathbb{E} \left[\frac{1}{\vec{x}^T \vec{x}} | x_i = 1, x_j = 1 \right] \geq \frac{1}{\mathbb{E} \left[\vec{x}^T \vec{x} | x_i = 1, x_j = 1 \right]}. \tag{5.16}$$

Furthermore, for any i, j,

$$\mathbb{E} \left[\vec{x}^T \vec{x} | x_i = 1, x_j = 1 \right] \leq 2 + \sqrt{n-2}. \tag{5.17}$$

To see this, observe that $\mathbb{E} \left[\vec{x}^T \vec{x} \right] = \|\vec{v}\|_1 \leq \sqrt{n}$. So we have

$$\mathbb{E} \left[\frac{1}{\vec{x}^T \vec{x}} | x_i = 1, x_j = 1 \right] \geq \frac{1}{2 + \sqrt{n-2}}. \tag{5.18}$$

Therefore,

$$\mathbb{E} \left[\frac{\vec{x}^T A \vec{x}}{\vec{x}^T \vec{x}} \right] = \sum_{i \neq j} A_{ij} v_i v_j \mathbb{E} \left[\frac{1}{\vec{x}^T \vec{x}} | x_i = 1, x_j = 1 \right] \tag{5.19}$$

$$\geq \sum_{i \neq j} A_{ij} v_i v_j \frac{1}{2 + \sqrt{n-2}} = \frac{\lambda_1}{2 + \sqrt{n-2}}. \tag{5.20}$$

That is,

$$O(\sqrt{n}) \mathbb{E} \left[\frac{\vec{x}^T A \vec{x}}{\vec{x}^T \vec{x}} \right] \geq \lambda_1 \geq OPT. \tag{5.21}$$

□

In Section 5.3 we show that this result is tight.

5.2.1 Enhancements for practical use

When using these algorithms to analyze real-world networks in practical applications, it might be beneficial to apply tweaks to enhance their flexibility and produce a wider variety of results. We propose the following simple enhancements.

EIGENSIGN: As discussed above, EIGENSIGN always outputs a solution involving all the vertices in the network. We can circumvent this shortcoming by including only those vertices such that the corresponding entry of the eigenvector $\vec{v}$ is at least a user-defined threshold τ. That is, $x_i = sgn(v_i)$ if $|v_i| \geq \tau$, 0 otherwise.

Random-Eigensign: The $\sqrt{n}$-approximation guaranteed by Random-Eigensign is matched in the extreme case in which all entries of the eigenvector $\vec{v}$ are of equal magnitude. Paradoxically, in this situation a solution comprised of all vertices would be optimal, but each vertex is included with a small probability of $1/\sqrt{n}$. We could of course fix this by modifying the probabilities to be $\min\{1, \sqrt{n}|v_i|\}$ for each i. However, in the opposite extreme, where most of the magnitude of $\vec{v}$ is concentrated in one entry, modifying the probabilities this way might disproportionately boost the likelihood of including undesirable vertices. An adequate multiplicative factor for both cases is $\|\vec{v}\|_1$, modifying the probabilities to be $\min\{1, \|\vec{v}\|_1|v_i|\}$ for each i; in the first case, all vertices are taken with probability 1, while in the second, the probabilities remain almost unchanged. We employed this factor in our experiments with satisfactory results.

An obvious question arising is whether the approximation guarantee of Random-Eigensign could be improved using the modification described above. This question is left for future investigation.

5.3 Hardness and tightness

5.3.1 Hardness (Theorem 5.1)

In this section, we refer to a solution of 2PC as S_1, S_2, which denote the subsets of vertices that are assigned a 1 or a -1, respectively, in the solution vector $\vec{x}$. Given a vertex $v \in V$ and a subset of vertices $S \subseteq V$, we use $d_+(v, S)$ (respectively $d_-(v, S)$) to denote the number of '+' edges (respectively '−' edges) connecting v to other vertex in S.

We exploit the following result in our proof. It can be easily verified by examining the behavior of the cost functions when moving one vertex from one set to the other, so we omit the proof.

Proposition 5.4. *If we require $S_1 \cup S_2 = V$, problem* 2CC-Full *is equivalent to* 2CC*, i.e., their optimal solutions are the same.*

We now prove that 2PC is **NP**-hard by reduction from 2CC, which has been shown to be **NP**-hard by Shamir et al. [211].

Proof of Theorem 5.1. Given a graph $\tilde{G} = (\tilde{V}, \tilde{E})$ with n vertices as instance of 2CC, we construct a graph $G = (V, E)$ to be an instance of 2PC as follows. For every vertex in $\tilde{V}$ we create a corresponding vertex in V, and for every edge in $\tilde{E}$ we add an edge in E between the corresponding vertices in $\tilde{V}$, and having the same sign. Furthermore, for every vertex v in $\tilde{V}$ we introduce a clique of $m > 3n$ vertices (and positive edges) and a '+' edge between v and every vertex in the clique. The strategy to prove hardness is the following. We first restrict ourselves to *complete* solutions of 2PC (i.e. $S_1 \cup S_2 = V$), which can of course be mapped to solutions of 2CC. We prove that if one such complete solution S_1, S_2 optimizes 2PC, the corresponding solution of 2CC is also a maximizer. Second, we show that any optimal solution of the constructed instance of 2PC is complete.

We denote the objective of the problems 2CC and 2PC, on instances $\tilde{G}$ and G, by W_{2CC} and W_{HPC}, respectively. We consider a solution $\tilde{S}_1, \tilde{S}_2$ of 2CC, and a solution S_1, S_2 of 2PC, such that $\tilde{S}_1 \subseteq S_1$ and $\tilde{S}_2 \subseteq S_2$. Let us first restrict our attention to complete solutions of 2PC. Observe that

$$\begin{aligned} W_{HPC}(S_1, S_2) = &\frac{1}{n + nm}\left(W_{2CC}(\tilde{S}_1, \tilde{S}_2) - D(\tilde{S}_1, \tilde{S}_2)\right) \\ &+ \frac{1}{n + nm}\left(|S_1|m + |S_2|m + n\binom{m}{2}\right), \end{aligned} \tag{5.22}$$

where $D(\tilde{S}_1, \tilde{S}_2) = \sum_{v \in \tilde{S}_1} d_+(v, \tilde{S}_2) + \sum_{v \in \tilde{S}_2} d_+(v, \tilde{S}_2) + d_-(v, \tilde{S}_1) + d_-(v, \tilde{S}_2)$, that is, the sum of disagreements in the resulting clustering. Note that $W_{2CC}(\tilde{S}_1, \tilde{S}_2) - D(\tilde{S}_1, \tilde{S}_2)$ is exactly the objective of the 2CC-Full problem. In other words, the obective of 2PC on G is proportional to the objective of 2CC-Full on $\tilde{G}$ plus a constant. By Proposition 5.4, the first part of the proof is complete.

We now consider a complete solution S_1, S_2 and show that removing vertices leads to no further improvement. Suppose we remove a set R of r vertices from the solution. We want to show

$$\frac{\nu(W_{HPC}(S_1,S_2)) - \Delta(R)}{n+nm-r} < \frac{\nu(W_{HPC}(S_1,S_2))}{n+nm}, \tag{5.23}$$

where $\nu(W_{HPC}(S_1,S_2))$ is the numerator of $W_{HPC}(S_1,S_2)$, and $\Delta(R)$ is the net change after removing the vertices in R (i.e., the number of agreements minus disagreements that are removed). Equivalently, we want to show $\Delta(R)(n+nm) > r\nu(W_{HPC}(S_1,S_2))$. We first consider that the removed vertices are in $\tilde{V}$. Observe that

$$\Delta(R) \geq rm - \binom{r}{2} - r(n-r), \tag{5.24}$$

$$\nu(W_{HPC}(S_1,S_2)) \leq \binom{n}{2} + nm + n\binom{m}{2}. \tag{5.25}$$

This upper bound holds because the right hand side simply counts all possible '+' and '−' edges, the edges between each actual vertex and its clique, and the edges within cliques. It is therefore sufficient to show

$$rm - rn + r^2 - \binom{r}{2} > r\frac{\binom{n}{2} + nm + n\binom{m}{2}}{n+nm}. \tag{5.26}$$

After some manipulations and relaxing the condition to remove the dependence on r, we arrive at the following sufficient condition:

$$(m-n)(n+nm) > \binom{n}{2} + nm + n\binom{m}{2}, \tag{5.27}$$

which holds for $m > 3n$. The case where the removed vertices are not in $\tilde{V}$ can be analyzed in the same manner. We have shown that we can reduce an instance of 2CC to a polynomially-sized instance of 2PC. □

5.3.2 Tight example for Random-Eigensign.

We consider a complete graph where all edges are positive, except for one Hamiltonian cycle comprised of negative edges. Without loss of generality, we can order the vertices so that the adjacency matrix is

$$A = \begin{pmatrix} 0 & -1 & 1 & \dots & 1 & -1 \\ -1 & 0 & -1 & 1 & \dots & 1 \\ 1 & -1 & 0 & -1 & 1 & \dots \\ & & \vdots & \vdots & & \\ 1 & 1 & \dots & -1 & 0 & -1 \\ -1 & 1 & \dots & 1 & -1 & 0\dots \end{pmatrix}.$$

That is, matrix A is comprised entirely of ones, save for the subdiagonal and superdiagonal entries, which are -1, and $A_{n1} = A_{1n} = -1$. Note that a constant vector $\vec{v}$, i.e., satisfying $v_i = v_j$ is an eigenvector of eigenvalue $n-5$. Since $\sum_i \lambda_i^2(A) = \|A\|_F^2 = n(n-1)$, the eigenvalue $n-5$ will be the largest if

$$\frac{n(n-1)}{2} < (n-5)^2, \tag{5.28}$$

which holds for $n > 16$. Note that $\sqrt{n}\vec{v}$ is a feasible solution for 2PC. We now show that Random-Eigensign attains a value of $\Theta(\sqrt{n})$.

We first rely on Equality (5.11) to obtain the following:

$$\begin{aligned}
&\mathbb{E}\left[\frac{\vec{x}^T A\vec{x}}{\vec{x}^T\vec{x}}\right] \\
&= \sum_{k=1}^{n}\frac{1}{k}\sum_{i\neq j} A_{ij}s_{ij}Pr(x_i=1,x_j=1)Pr(\vec{x}^T\vec{x}=k|x_i=1,x_j=1) && (5.29)\\
&= \sum_{k=1}^{n}\frac{1}{k}\sum_{i\neq j} A_{ij}v_iv_jPr(\vec{x}^T\vec{x}=k|x_i=1,x_j=1) && (5.30)
\end{aligned}$$

Now, observe that given k, $Pr(\vec{x}^T\vec{x}=k|x_h=1,x_l=1)$ is constant for all $i\neq j$. Thus, for arbitrary h,l,

$$\begin{aligned}
\mathbb{E}\left[\frac{\vec{x}^T A\vec{x}}{\vec{x}^T\vec{x}}\right] &= \sum_{k=1}^{n}\frac{1}{k}Pr(\vec{x}^T\vec{x}=k|x_h=1,x_l=1)\sum_{i\neq j}A_{ij}v_iv_j && (5.31)\\
&= (n-5)\mathbb{E}\left[\frac{1}{\vec{x}^T\vec{x}}\middle| x_h=1,x_l=1\right]. && (5.32)
\end{aligned}$$

Observe that when all entries of $\vec{v}$ are equal in absolute value, $\vec{x}^T\vec{x}$ is a binomial variable with parameters $(n,|v_i|)=(n,1/\sqrt{n})$. Thus, by Jensen's inequality we have

$$\mathbb{E}\left[\frac{1}{\vec{x}^T\vec{x}}\middle| x_h=1,x_l=1\right] \geq \frac{1}{\mathbb{E}\left[\vec{x}^T\vec{x}|x_h=1,x_l=1\right]} = \Omega(1/\sqrt{n}). \tag{5.33}$$

Furthermore, it is known [63] that

$$\mathbb{E}\left[\frac{1}{\vec{x}^T\vec{x}}\middle| x_h=1,x_l=1\right] = \mathcal{O}(1/\sqrt{n}). \tag{5.34}$$

That is,

$$\mathbb{E}\left[\frac{1}{\vec{x}^T\vec{x}}\middle| x_h=1,x_l=1\right] = \Theta(1/\sqrt{n}). \tag{5.35}$$

Combining this with Equality (5.31) we get

$$\mathbb{E}\left[\frac{\vec{x}^T A\vec{x}}{\vec{x}^T\vec{x}}\right] = \Theta(\sqrt{n}) = \Theta\left(\frac{OPT}{\sqrt{n}}\right). \tag{5.36}$$

5.4 Experiments

This section presents the evaluation of the proposed algorithms: first (Section 5.4.1) we present a characterization of the polarized communities discovered by our methods; then (Section 5.4.2) we compare our methods against non-trivial baselines in terms of objective, efficiency and scalability, and ability to detect ground-truth planted polarized communities in synthetic datasets. Finally, we show a case study about political debates (Section 5.4.3).

Datasets. We select publicly-available real-world signed networks, whose main characteristics are summarized in Table 5.1. HighlandTribes[1] represents the alliance structure of the Gahuku–Gama tribes of New Guinea. Cloister[1] contains the esteem/disesteem relations of monks living in a cloister in New England (USA). Congress[1] reports (un/)favorable mentions of politicians speaking in the US Congress. Bitcoin[2] and Epinions[2] are who-trusts-whom networks of the users of Bitcoin OTC and Epinions, respectively. WikiElections[1] includes the votes about admin elections of the users of the English Wikipedia. Referendum[3] [153] records Twitter data about the 2016 Italian Referendum: an interaction is negative if two users are classified with different stances, and is positive

[1]konect.cc

[2]snap.stanford.edu

[3]researchgate.net/publication/324517807_Annotated_Corpus_for_Stance_Detection_-_Italian_Constitutional_Referendum_2016

Table 5.1: Signed networks used: number of vertices and edges; ratio of negative edges ($\rho_- = \frac{|E_-|}{|E_+ \cup E_-|}$); L_1-norm of the eigenvector corresponding the largest eigenvalue of A ($\|\vec{v}\|_1$); and, ratio of non-zero elements of A ($\delta = \frac{2|E_+ \cup E_-|}{|V|(|V|-1)}$).

Real-world datasets	$\lvert V\rvert$	$\lvert E_+ \cup E_-\rvert$	ρ_-	$\|\vec{v}\|_1$	δ
HighlandTribes	16	58	0.50	3.61	0.48
Cloister	18	125	0.55	3.71	0.81
Congress	219	521	0.20	10.51	0.02
Bitcoin	5 k	21 k	0.15	31.21	$1.2e{-}03$
WikiElections	7 k	100 k	0.22	35.96	$3.9e{-}03$
Referendum	10 k	251 k	0.05	42.66	$4.2e{-}03$
Slashdot	82 k	500 k	0.23	59.46	$1.4e{-}04$
WikiConflict	116 k	2 M	0.62	119.66	$2.9e{-}04$
Epinions	131 k	711 k	0.17	72.20	$8.2e{-}05$
WikiPolitics	138 k	715 k	0.12	91.48	$7.4e{-}05$
WikiConflict16$\lvert V\rvert$	1 M	67 M	0.62	129.04	$3.4e{-}05$
Epinions16$\lvert V\rvert$	2 M	23 M	0.17	75.99	$9.5e{-}06$

otherwise. Slashdot[2] contains friend/foe links between the users of Slashdot. The edges of WikiConflict[2] represent positive and negative edit conflicts between the users of the English Wikipedia. WikiPolitics[1] represents interpreted interactions between the users of the English Wikipedia that have edited pages about politics.
In order to study scalability, we artificially augment two of the largest datasets to produce networks with millions of vertices and tens of millions of edges (details in Section 5.4.2).

Implementation. All methods, with the exception of algorithm FOCG (details in Section 5.4.2), are implemented in Python (v. 2.7.15) and compiled by Cython. The experiments run on a machine equipped with Intel Xeon CPU at 2.1GHz (32 cores) and 128GB RAM.[4]

5.4.1 Solutions characterization

We first characterize the solutions discovered by our methods Eigensign (for short E) and Random-Eigensign (RE), and we show how the tweaks described in Section 5.2.1 enhance their flexibility in producing a wider variety of results. In particular, algorithm E evaluates the threshold τ for each $|v_i|$ discretized at the third decimal digit. This operation is carried out efficiently, since $\vec{v}$ is computed only once regardless of the number of evaluated values of τ. On the other hand, algorithm RE employs $\|v\|_1$ as multiplicative factor, therefore the probabilities are modified to be $\min\{1, \|\vec{v}\|_1 |v_i|\}$. In the following, we refer to the two communities included in the solutions as S_1 and S_2, namely the subsets of vertices that are assigned with 1 and -1, respectively, by the solution vector $\vec{x}$.

Figure 5.2 shows how the solutions returned by algorithm E are affected by parameter τ in terms of polarity, edge-agreement ratio (i.e., the portion of edges in the solution that comply with the polarized structure), and size on four datasets. In all of them, the three measures follow very similar trends. The highest polarity is achieved at about a fourth of the domain of τ, when most of the neutral vertices are discarded. The edge-agreement ratio, instead, is consistently close or equal to 1: the solutions have a coherent polarized structure regardless of the chosen τ. Finally, as expected, the number of vertices included in the solutions decreases as τ grows, and presents a substantial decay at the beginning of the domain. Therefore, parameter τ is a powerful enhancement that allows algorithm E to be tuned to return the most suitable solution for the domain under analysis.

For algorithm RE, due to the randomness, we report the best solution with respect to polarity out of 100 runs. We do the same for the baseline LS, that we introduce in Section 5.4.2. Figure 5.3 shows the boxplots of the edge-agreement ratio over the larger datasets. It has significant values in

[4]Code and datasets available at github.com/egalimberti/polarized_communities.

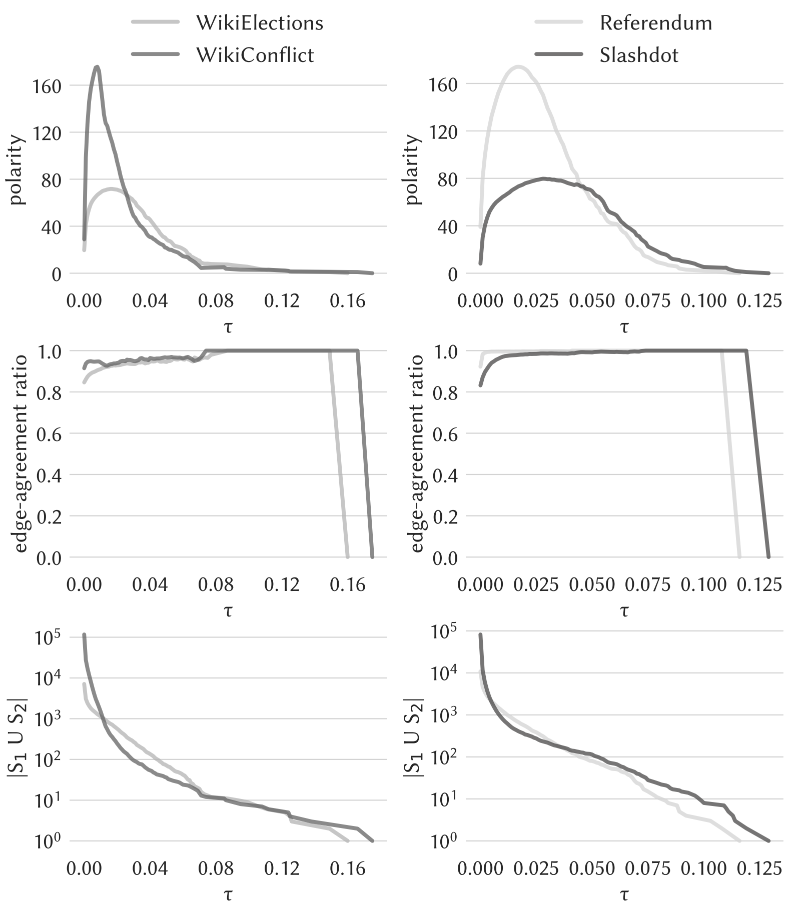


Figure 5.2: Solutions produced by E as a function of τ.

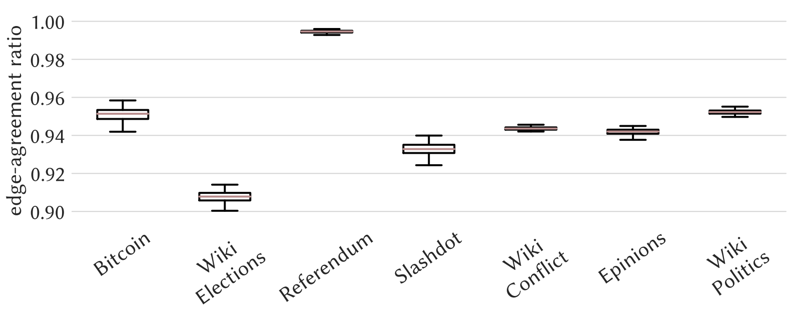


Figure 5.3: Edge-agreement ratio of the solutions of RE.

all cases, above 0.9, and is stable among the different executions. Polarity and solution size for all datasets are reported in Figure 5.4. For such measures, we do not show boxplots as they are highly dependent on the specific dataset and very stable over different runs: their index of dispersion is lower than 0.01 and $3.2e-05$, respectively, for all datasets. This confirms that algorithm RE is very stable and does not require multiple executions to identify high-quality solutions.

5.4.2 Comparative evaluation

We next compare algorithms E and RE against non-trivial baselines inspired by methods proposed in the literature for different yet related problems.

FOCG. The first method we compare to, whose objective is to find k *oppositive cohesive groups*

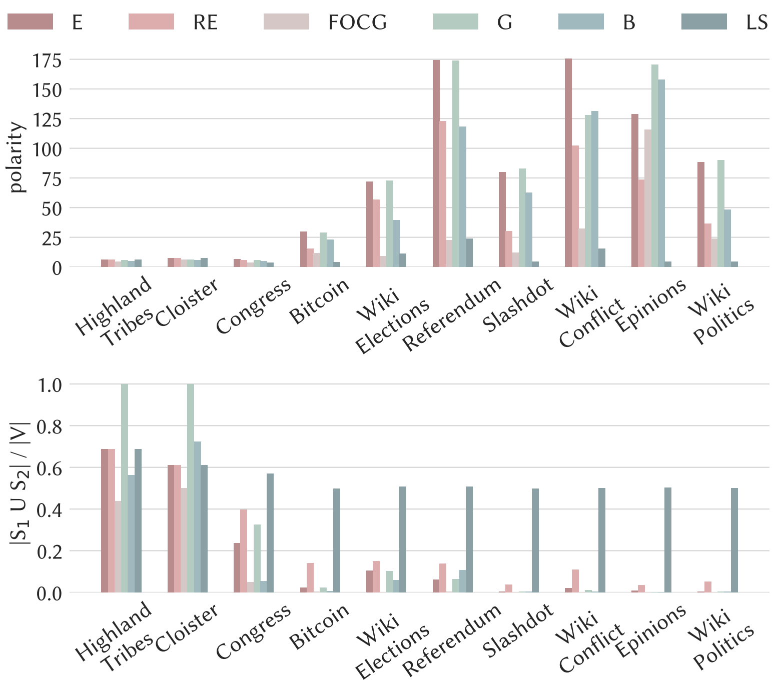


Figure 5.4: Polarity and solution size (normalized) of the proposed algorithms and baselines.

(i.e., k-OCG) in signed networks, is taken from [59]. Algorithm FOCG detects p different k-OCG structures within the input signed network, among which we elect the one having highest polarity as the ultimate solution to our problem. We setup the algorithm with the default configuration (i.e., $\alpha = 0.3$ and $\beta = 50$) and $k = 2$. The code is provided by the authors.

Greedy. Our second baseline is inspired by the 2-approximation algorithm for *densest subgraph* [55]. Algorithm GREEDY (for short G), iteratively removes the vertex minimizing the difference between the number of positive adjacent edges and the number of negative adjacent edges, up to when the graph is empty. At the end, it returns the subgraph having the highest polarity among all subgraphs visited during its execution. The assignment of the vertices to the clusters is guided by the sign of the components of the eigenvector $\vec{v}$, corresponding to the largest eigenvalue of A.

Bansal. A different approach, motivated by the strong similarity to our setting, is inspired by Bansal's 3-approximation algorithm for 2CC on complete signed graphs [24]. For each vertex $u \in V$, this algorithm, which we refer to as BANSAL (for short B), identifies u together with the vertices sharing a positive edge with u as one cluster, and the vertices sharing a negative edge as the other. Of these $|V|$ possible solutions, it returns the one maximizing polarity.

LocalSearch. Finally, we consider a local search approach (LOCALSEARCH, for short LS), guided by our objective function. Algorithm LS starts from a set of vertices chosen at random; at each iteration, it adds (removes) to (from) the current solution the vertex that maximizes the gain in terms of polarity, and finally terminates when the gain of moving any vertex is lower than 0.2. Also for this algorithm, the assignment of the vertices to the clusters is guided by the signs of $\vec{v}$.

Note that algorithms E, RE, and FOCG exploit the multi-core architecture (all 32 cores), while the other baselines run on a single core except for the computation of polarity which is implemented to use multiple cores.

Figure 5.4 reports the achieved values of polarity for all compared algorithms on all datasets, as well as the size (normalized by $|V|$) of the solutions returned. In most of the cases, algorithm E results the be the most competitive method with respect to polarity; on the other hand, algorithm RE is able to return solutions of high polarity for the small-sized datasets. Algorithm FOCG is instead not competitive since its solutions are of extremely small size (note that the numerator of our objective can be up to quadratic in the size of the denominator, so size matters for reaching

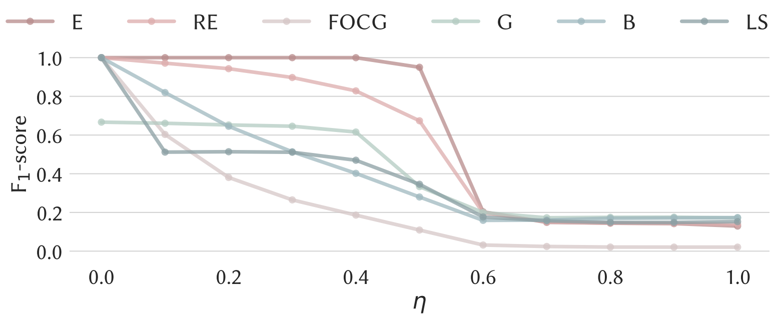


Figure 5.5: F_1-score as a function of the noise parameter η ($n_c = 100$, $n_n = 800$).

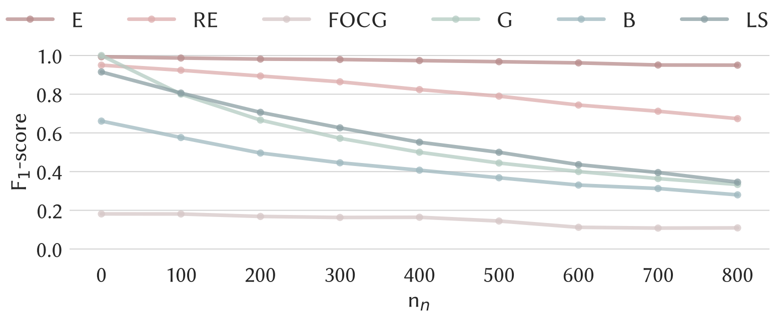


Figure 5.6: F_1-score as a function of the number noisy vertices n_n ($n_c = 100$, $\eta = 0.5$).

high polarity). Algorithm G has, in general, polarity comparable to algorithm E, slightly higher in a few cases (with the exception of WikiConflict, in which algorithm E clearly outperforms algorithm G). However, it must be noted that algorithm G often returns a very dense subgraph as one of the two communities, leaving the second community totally empty, which is, of course, undesirable in our context. Algorithms B and LS, instead, exhibit weak performance in terms of polarity: their search spaces strongly depend on the neighborhood structure of the vertices (for B), or on the random starting sets (for LS). About the solution size, all methods, with the exception of algorithms FOCG and LS, return solutions of reasonable dimension with respect to the number of vertices of the networks. Excluding the small empirical datasets (i.e., HighlandTribes, Cloister, and Congress), the size of the solutions is below 20% of the input.

Planted polarized communities. In order to better assess the effectiveness of the various algorithms, we test their ability to detect a known planted solution, concealed within varying amounts of noise. For our purposes we create a collection of synthetic signed networks identified by three parameters: the size of each planted polarized community $n_c = |S_1| = |S_2|$ (for convenience, we consider communities having the same size); the number of noisy vertices external to the two polarized communities $n_n = |V \setminus (S_1 \cup S_2)|$; and, a *noise parameter* $\eta \in [0, 1]$ governing the edge density and agreement to the model. In detail:

- edges inside S_1 (respect. S_2) exist and are positive with probability $1 - \eta$, exist and are negative with probability $\eta/2$, and do not exist with probability $\eta/2$;
- edges between S_1 and S_2 exist and are negative with probability $1 - \eta$, exist and are positive with probability $\eta/2$, and do not exist with probability $\eta/2$;
- all other edges (outside the two polarized communities) exist with probability η and have equal probability of being positive or negative.

The higher η, the less internally dense and polarized the two communities are, and the more connected the noisy vertices are, both between themselves and to the communities. Observe how the case with no noise ($\eta = 0$) corresponds to the "perfect" structure.

For each configuration of the parameters, we create 10 different networks and we report the average F_1-score in detecting which vertices belong to S_1 (respect. S_2) and which ones to $V \setminus (S_1 \cup$

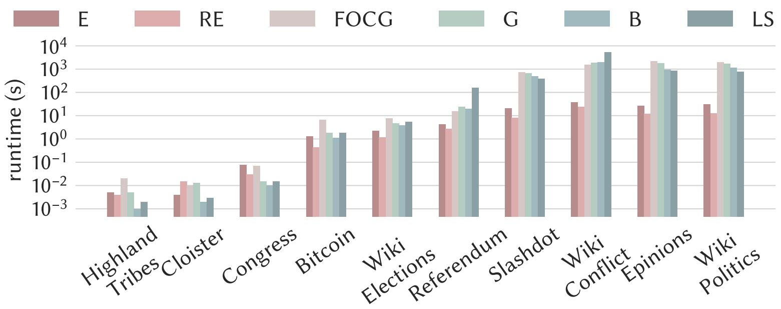


Figure 5.7: Runtime of the proposed algorithms and baselines.

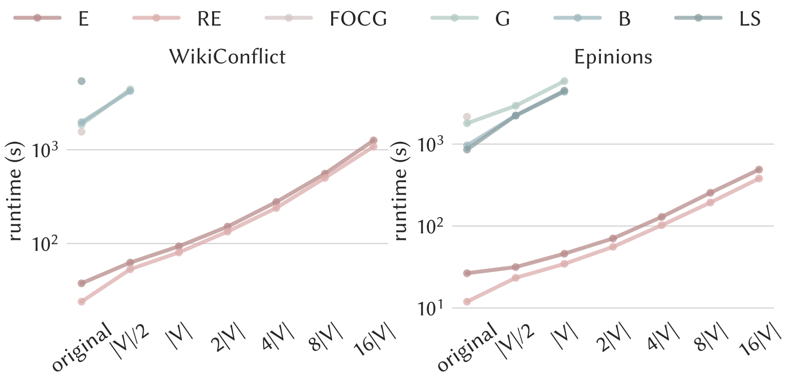


Figure 5.8: Scalability: runtime of the proposed algorithms and baselines as a function of the number of injected dummy vertices, for WikiConflict and Epinions.

S_2)[5].

In Figure 5.5 we fix the size of the synthetic network to 1 000 ($n_c = 100$, $n_n = 800$) and vary η. For $\eta = 0$, all algorithms have, as expected, maximum F_1-score with the exception of algorithm G that, even in the case without noise, is not able to exactly identify the planted structure. As expected, as η increases, the F_1-score decays for all methods; however, our algorithms E and RE clearly outperform the others. Figure 5.6 shows the F_1-score varying the number n_n of vertices external to the polarized communities, with fixed $n_c = 100$ and $\eta = 0.5$. Again algorithms E and RE stand out, especially E that presents F_1-score close to the maximum in all cases. Algorithm FOCG has the poorest performance: the small size of its solutions penalizes the recall, which is never greater than 0.1.

Runtime and scalability. Figure 5.7 reports the runtime of all algorithms over all datasets. Algorithms E and RE, with their practical enhancements discussed in Section 5.2.1, always terminate in less than 40 seconds. The runtime of the baselines is instead more than an order of magnitude higher than algorithms E and RE.

In order to assess the scalability of our methods, we augment two of the larger datasets (i.e., WikiConflict and Epinions) by artificially injecting dummy vertices having a number of randomly-connected edges equal to the average degree of the original network, while maintaining ρ_- (i.e., the ratio of negative edges). The largest datasets created in this way contain up to 2 M vertices and 67 M edges (see Table 5.1 for details). Note that, as the quantity of noise increases, δ, i.e., the ratio of non-zero elements of the adjacency matrix, decreases. Nonetheless, δ differs with respect to the original datasets less than an order of magnitude in both cases, making the following results about scalability significant.

Figure 5.8, which reports on the x-axis the number of dummy vertices added, shows that the runtime of both algorithms E and RE grows linearly with the number of vertices. Among the two, algorithm E is slightly slower than algorithm RE due to the evaluation of multiple values of the

[5]For instance *recall* is defined as $(|S_1^* \cap S_1| + |S_2^* \cap S_2|)/|S_1 \cup S_2|$, where S_1^* (S_2^*) denotes the first (second) community returned by the algorithm while S_1 (S_2) denotes the corresponding ground-truth one.

threshold τ. In the worst case, algorithms E returns in about 21 minutes. On the other hand, the baselines cannot complete each computation within the 10 000 seconds timeout that we apply. No baseline terminates for more than $|V|$ additional dummy vertices on both datasets. In particular, algorithm FOCG is able to handle in reasonable time only the original versions, with no dummy vertices. It should be noted that algorithm FOCG recursively finds a polarized structure, removes the corresponding subgraph, and repeats the process on the remaining vertices. While each one of these iterations runs efficiently, most of the found structures are too small to be of interest in our setting. Thus, it is necessary to allow the algorithm to complete many of such iterations in order to find interesting solutions.

5.4.3 Case study: political debate

We finally analyze the solution extracted by algorithm RE from Referendum to show tangible benefits of our problem formulation and algorithms in identifying the two most polarized communities in a signed network modeling political debates. The Referendum dataset includes Twitter data about the Italian Constitutional Referendum held on December 4, 2016 (more information about the Referendum can be found at this link). The original data seed consists of about 1 M tweets posted between November 24 and December 7, 2016, extended by collecting retweets, quotes, and replies. The users (10 884 in total) are annotated with a stance about their outlook towards the Referendum as favorable (5 137), against (1 510), or none (4 237) when the stance cannot be inferred. An interaction (edge) is considered negative if occurred between two users (vertices) of different stances, and is positive otherwise, i.e., we treat "none" users as neutral, in agreement with both favorable and against users.

The solution output by algorithm RE consists of two communities of 27 and 1 558 users, accounting for 14% of the overall user set. Both communities have more than 99% of positive edges within and 74% of negative edges in-between, and thus, are highly polarized. Interestingly, all the 27 users of the smaller community are classified as favorable to the Referendum, while the users in the larger community as against (75%) or "none" (24%), with the exception of 3 favorables. Moreover, the vertices in the solution have, on average, 183.12 adjacent edges compared to the average 22.85 contacts of the vertices outside, meaning that the solution identifies the "core" of the controversies, i.e., a set of intensely debating users about the Referendum. These results provide evidence of the practical value of our problem formulation and algorithms to identify two communities that are polarized about a certain topic.

5.5 Summary

Detecting extremely polarized communities might enable fine-grained analysis of controversy in social networks, as well as open the door to interventions aimed at reducing it [103]. As a step in this direction, in this chapter we introduce the 2-Polarized-Communities problem, which requires finding two communities (subsets of the network vertices) such that within communities there are mostly positive edges while across communities there are mostly negative edges. We prove that the proposed problem is **NP**-hard and devise two efficient algorithms with provable approximation guarantees. Through an extensive set of experiments on a wide variety of real-world networks, we show how the proposed objective function can be optimized to reveal polarized communities. Our experiments confirm that our algorithms are more accurate, faster, and more scalable than non-trivial baselines.

Part II

Applications

Chapter 6

Measures and patterns of the scientific migration network

Human migration is a phenomenon of crucial importance in modern history that radically evolves over time, is affected by historical and economical events, and is rooted in the alliance system of the countries. It is known for shaping local demographics, politics, and regulations; and, also, for influencing global wealth and world-wide society [185]. In recent years, human migration has become elder and is likely to increase even more in the next decades, leaving huge implications in both origin and destination countries of the migrants [146]. The definitive outcome of human migration is subtle and extremely unpredictable, especially on the long term, due to the need of addressing different borders: geographical, political, and even cultural [196]. For these reasons, human migration is perceived in many different manners and, consequently, treated by local states with opposite aims: it is sometimes encouraged, rather discouraged [207].

Knowledge, ideas, and information are considered to be among the major economic production factors in today's economy and are naturally embedded in researchers, scientists, and academics who, through their migrations, move such precious good from a location to another [177]. On the long term, the international scientific migration could impact fundamental socio-economic aspects of the countries, such as scientific, technological, and productive assets [193]. Albeit, most of the times, this phenomenon lacks the urgency of survival, it is highly competitive in terms of choice of the destination countries, as pointed out in [68]. Moreover, the international scientific migration is expected to be develop faster (compared to the general human migration), since the permanence in a visiting country can be considered a structural part of the majority of the academic careers and it is often short-term.

In this chapter we show how it is possible to study the international migration of researchers, scientists, and academics by means of complex-network analysis for identifying measures and patterns that describe the countries having a central role in such phenomenon. The data we employ in this study were collected from 2.8 millions public profiles of ORCID [43], a growing platform dedicated to researchers. Given its nature, the (scientific) migration can be modeled by means of a network that we define to be temporal, weighted, and directed. In particular, nodes represent world countries and edges account for a migratory flow from a country to another. Edge weights stand for the size of the migratory flow in terms of migrants, while timestamps represents years from 2000 to 2016. We name such network *scientific migration network* (*SMN* for short).

In our setting, a country is established as central in the scientific migration process if it is able to provide or attract a large number of outcoming or incoming researchers. Certainly, these are two antithetical aspects that worth to account separately and from a global perspective. To purse such objective, we employ the well-known *weighted hyperlink-induced topic search* (HITS) algorithm [143] on the scientific migration network to identify *hubs* and *authorities*. We compare the results obtained by HITS to other local and global methods, which fail to unveil the interplay between exporting and importing researchers on large scale. Further, we investigate the local patterns and characteristics of successors of hubs and predecessors of authorities to derive the motivations behind the HITS algorithm. Finally, we showcase how to employ network visualization to evince the temporal evolutions of such patterns/characteristics of selected hubs and authorities.

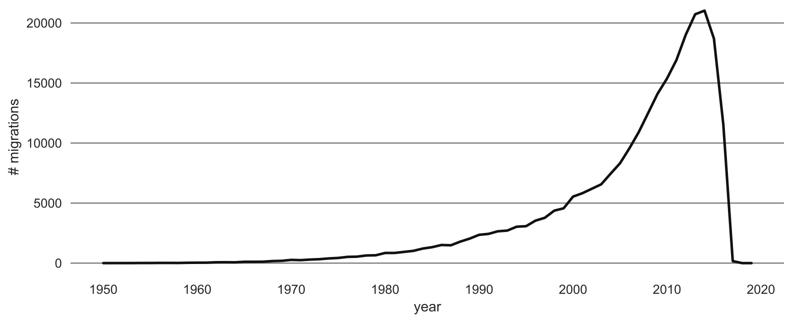


Figure 6.1: Distribution of the number of ORCID members migrating per year, from 1950 to 2020.

Our results show high correlation between hub and authority countries. In particular, we are able to identify a set of actors that occupies a privileged position in the scientific migration network, being both important hubs and central authorities, since they are able to attract researchers and, at the same time, to provide scientist to the most prestigious states. Moreover, the majority of the central countries in the scientific migration network shares similar characteristics/patterns of their local neighborhood/cluster, i.e., they provide/receive scientist to/from many different states instead of having a few well-established migration corridors. Also, we observe various patterns that lead actors with similar hub or authority score to occupy different positions in the community structure of the scientific migration network. External factors, e.g., regulations, political alliances, investments in research, development, and education, are expected to play an important role in such results and to add an additional layer of complexity that deserves to be further investigated.

By this work, we provide the following contributions:

- using ORCID public profiles as data source, we model the scientific migration phenomenon by means of a temporal weighted directed network (Section 6.1);
- we employ the weighted hyperlink-induced topic search algorithm to identify hubs and authorities of the scientific migration network and compare it with other local and global approaches (Section 6.2);
- we characterize the local patterns and characteristics of successors of hubs and predecessors of authorities to derive the motivations behind the HITS algorithm (Section 6.3);
- by means of network visualization, we show how to evince the temporal evolution of the local patterns/characteristics of selected hubs and authorities (Section 6.4).

6.1 Scientific migration network

6.1.1 Dataset

The dataset employed in this work has been assembled by Bohannon and Doran [43] through the gathering of 2.8 millions ORCID public profiles. ORCID is a nonprofit organization that collects contributions, affiliations, and personal information of the subscribed researchers. Given the affiliation history of each member, we are able to identify the location, in terms of country, of his/her workplace over time and infer his/her migration across different states in time. In the following, we study the dataset on annual basis due to data limitations, i.e., the temporal information input by the users often lacks of the month granularity, and to ease of interpretation.

Figure 6.1 shows the distribution of the number of migrations, i.e., the number of ORCID members that changed the country they worked in, per year, from 1950 to 2020. Most of the data is concentrated in the 21st century, with a peak in 2014. The decay of recorded migrations after 2014 might be due to temporal bias given by the time when the dataset was gathered, i.e., in 2017. Even if ORCID was founded in 2012, members are allowed to insert information about their previous occupations and their future ones; then, migrations appear before 2012 and after 2017.

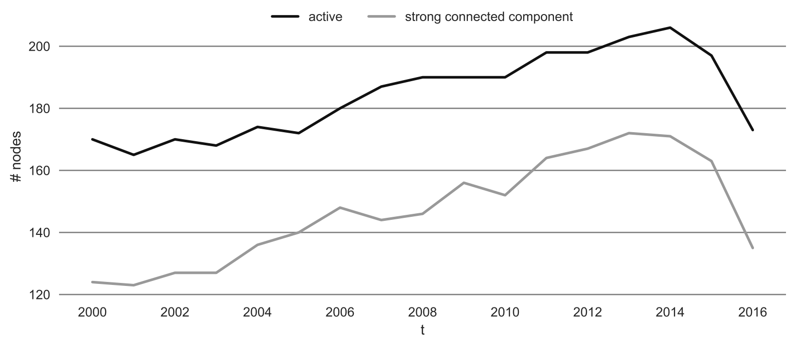


Figure 6.2: Evolution of the number of active nodes and the size of the strong connected component of the scientific migration network.

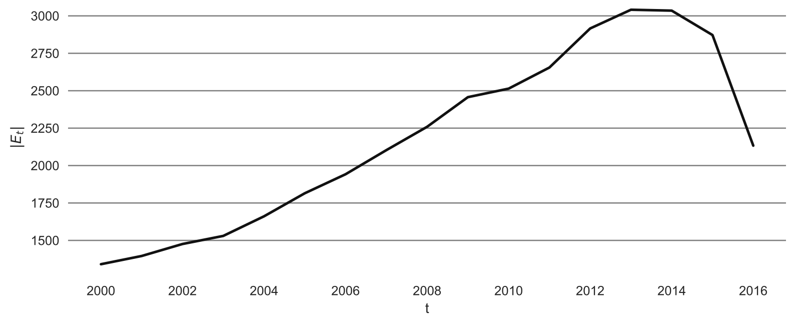


Figure 6.3: Evolution of the number of existing edges of the scientific migration network.

In their work, Bohannon and Doran [43] highlight that ORCID was not born with the specific aim of tracking researchers' movement. Therefore, the data we consider has structural limitations as well as biases. First of all, much of the information input by the members is retroactive since it is previous to the launch of ORCID in 2012. As a consequence, some of countries that nowadays have disappeared are present in the dataset, making the set of states considered for each year vary. Secondly, since its appearance, ORCID has always skewed towards younger researchers. In fact, members of recent Ph.D. are overrepresented in the dataset, reflecting the fact that younger researchers sign-up to ORCID more frequently than older ones. Finally, there are countries that are not fairly represented, namely, the distribution of the number of researchers per country does not follow the distribution of the overall population. Bohannon and Doran compare ORCID data in 2013 about scientific migrations to the UNESCO Science Report[1] to discover which countries are misrepresented; e.g., China, Russia, and Japan result to be underrepresented while, e.g, Spain, and Portugal are overrepresented. All in all, for these reasons, we cannot regard the dataset as a definitive picture of the scientific migrations. Nevertheless, we can exploit it to detect regularities and patterns by the construction of a network model, useful in the understanding of the global perspective of the phenomenon.

6.1.2 Network model

We consider a weighted directed temporal network $G = (V, T, \varpi)$, where V is a set of nodes, $T = [t_0, t_1, \dots, t_{max}] \subseteq \mathbb{N}$ is a discrete time domain, and $\varpi : V \times V \times T \to \mathbb{N}$ is a function defining for each pair of nodes $i, j \in V$ and each timestamp $t \in T$ the weight of edge (i, j) at time t. In the following, we refer to the weight of edge (i, j) at time t as $w_{ij,t}$, and we consider it missing if $w_{ij,t} = 0$. Let $s^{in}_{i,t} = \sum_{j \in V} w_{ji,t}$ and $s^{out}_{i,t} = \sum_{j \in V} w_{ij,t}$ represent the in-strength and the out-strength of node $i \in V$ at time $t \in T$, respectively. We also denote by $E_t = \{(i, j) \mid \varpi(i, j, t) > 0\}$ the set of edges existing at time $t \in T$.Finally, let W_t be the weighted adjacency matrix of G at

[1]`https://en.unesco.org/node/252273`

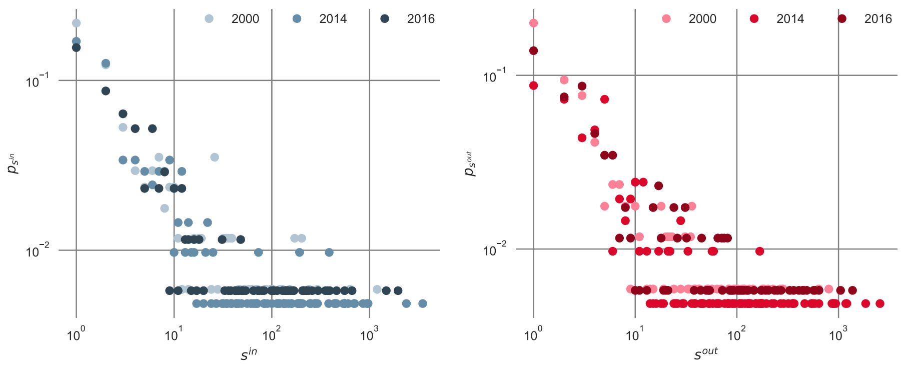


Figure 6.4: Distributions of the in-strength (left) and the out-strength (right) in the scientific migration network in 2000, 2014, and 2016.

time $t \in T$.

In our case, we identify the nodes of the network as the countries involved in the scientific migration process (231 in total), and an edge between two counties represents a migration route. Each edge between two nodes $i, j \in V$ is attributed with a time $t \in T$ and a weight w: a quartet (i, j, t, w) represents the migration of w researchers from country i to country j at time t. The time domain of the *scientific migration network* is $T = [2000, 2001, \ldots, 2016]$, composed of 17 years, since most of the data are concentrated between 2000 and 2016, and the geopolitical configuration of the countries is quite stable after 2000.

Figure 6.2 shows the number of active nodes (i.e., nodes $i \in V$ having $s^{in}_{i,t}$ and/or $s^{out}_{i,t}$ greater than 0) and the size of the strong connected component in the considered time domain T. Note that most of the nodes is active in 2014, which is also the year for which the dataset records the largest amount of information. Also the number of edges existing in each year follows a very similar trend (see Figure 6.3). For this reason, we consider year 2014 pivotal in the following analysis.

We report in Figure 6.4 the distributions of the in-strength and the out-strength in the scientific migration network in 2000, 2014, and 2016. The shapes of the distributions are very similar among the shown years, as well as the missing ones. Also, there are not notable differences between in-strength and out-strength. Such distribution will come in handy in the following, to create configuration models that preserve in-strength and out-strength sequences.

6.2 Hubs and authorities

6.2.1 A strength-based approach

A strength-based approach can be considered a straightforward attempt to numerically quantify the role of a country in the scientific migration network. We define the *drain index* of a country $i \in V$ at time $t \in T$ as

$$\beta(i,t) = \frac{s^{out}_{i,t} - s^{in}_{i,t}}{s^{out}_{i,t} + s^{in}_{i,t}}, \tag{6.1}$$

namely the number of outgoing researchers (i.e., out-strength) minus the number of incoming researchers (i.e., in-strength) normalized by their sum. It ranges from 1 to -1, where 1 indicates maximum brain drain (the country is a pure provider) while -1 means maximum brain gain (the country is a pure attractor). Values close to 0 are adopted by those countries having balanced values of out-strength and in-strength.

Figure 6.5 graphically shows the drain index for the year 2014, while Table 6.1 reports the ranking for specific countries: the five countries of highest β, the five countries of lowest β, and the five countries of highest out-strength. The countries standing out in Figure 6.5 are mainly located in Africa, southern Asia and in the Caribbean, while Europe and North America have milder colors. Considering the values in Table 6.1, it is easy to notice that extreme values of β are assigned when the number of migrations of a country is poor and completely unbalanced. For

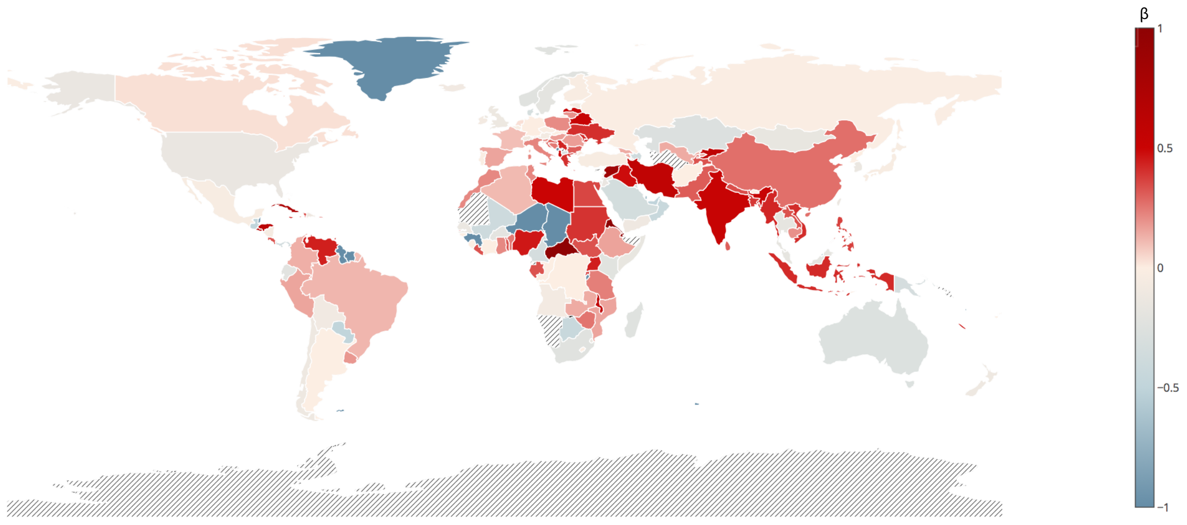


Figure 6.5: Drain index β in 2014. Countries without data are depicted in white with diagonal lines.

Table 6.1: Ranking (partial) of the countries by drain index β in 2014. For each country, out-strength and in-strength measured during such year are also reported. Countries highlighted in bold have the highest out-strength in 2014.

ranking	country	β	s^{out}	s^{in}
1	Sint Maarten	1.0	2	0
2	Eritrea	1.0	2	0
3	Central African Republic	1.0	1	0
4	Curacao	1.0	1	0
5	Saint Vincent and the Grenadines	1.0	1	0
85	**Spain**	0.03	80	74
90	**United Kingdom**	0.01	109	105
111	**France**	0.0	78	78
114	**United States**	−0.008	114	116
116	**Italy**	−0.01	71	73
202	Guinea	−1.0	0	2
203	Guyana	−1.0	0	2
204	Belize	−1.0	0	2
205	Niger	−1.0	0	3
206	Chad	−1.0	0	3

example, Sint Maarten has only two outgoing migrations, resulting in $\beta = 1$, while Chad has three incoming migrations and no outgoing researchers, then its β is -1. On the other hand, those countries playing a central role in the migration network have usually β close to 0 due to the high number of both outgoing and incoming researchers. This is the case of, e.g., the United Kingdom and the United States.

In order to favor the identification of the central countries in the migration process, we lift the network by removing the links having weight lower than a certain threshold tr. This operation has the aim of discarding weak and not meaningful interactions between countries. We experimentally verify $tr \in [1, 2, \ldots, 10]$, and we report part of the 2014 ranking in Table 6.2 for threshold values of 1 (original network), 2, and 3. Two important aspects have to be considered: (i) the extremes of the ranking are not robust with respect to the threshold (the rankings shown in Table 6.2 considerably differ for small variations of tr); (ii) even for low values of tr, a large portion of the network is neglected by the analysis (44% and 61% for $tr = 2$ and $tr = 3$, respectively). Therefore,

Table 6.2: Ranking (partial) of the countries by drain index β in 2014, varying the threshold tr. The five countries of highest β (ties broken by out-strength) and the five countries of lowest β (ties broken by in-strength) are reported.

ranking	$tr = 1$ (original)	$tr = 2$	$tr = 3$
1	Sint Maarten	Honduras	Syria
2	Eritrea	Barbados	Rwanda
3	Central African Republic	Bosnia and Herzegovina	Serbia
4	Curacao	South Sudan	Croatia
5	Saint Vincent and the Grenadines	Cambodia	Jamaica
202	Guinea	Burkina Faso	Macedonia (FYROM)
203	Guyana	Macedonia (FYROM)	Algeria
204	Belize	Madagascar	Botswana
205	Niger	Algeria	Mongolia
206	Chad	Botswana	Lithuania

we cannot consider this approach a reliable and fair analysis of the scientific migration network.

Additionally, we evaluate other strategies for normalizing the drain index by considering external data, such as the size of the overall population and the number of researchers of a country. Given the biases in the collected dataset, any normalization deriving from external sources would be inappropriate because it would misrepresent the results. Moreover, external data have to be temporal, at least of yearly granularity from 2000 to 2016, and available for all the countries included in the dataset. This is the case of the general population, but we cannot discover complete and coherent datasets about the size of the research population of all the studied states.

All in all, by this strength-based approach, we cannot induce strong conclusions nor provide a fair and robust analysis about the main characters of the scientific migration network. The main contributors to the network are not caught, and the removal of low-weight edges excessively limit the analyzed data. In addition, other normalizing strategies result to be unfeasible due to the nature of our data, or to the lack of complete external datasets. In order to overcome these limitations, we evaluate, in the following subsection, a method to asses the importance of the countries that takes into consideration the overall structure of the network and goes beyond the local strength structure of each node.

6.2.2 A global approach

A classic approach to assess the importance of a node in a network taking into account the global link structure is the well-known *PageRank* by L. Page et al. [187]. Let R_t be the PageRank matrix of $G = (V, T, \varpi)$ at time $t \in T$, defined as

$$r_{ij,t} = d\frac{w_{ij,t}}{\sum_{j\in V} w_{ij,t}} + (1-d)\frac{1}{|V|}, \tag{6.2}$$

where $d = 0.85$ is the dumpling factor. Note that, in this work, we consider the edge weights in the definition of R_t. The PageRank vector $\vec{r}_t = (r_{1,t}, \ldots, r_{|V|,t})^\intercal$ is obtained by repeating the iteration

$$\vec{r}_t(x+1) = R_t^\intercal \vec{r}_t(x) \tag{6.3}$$

until convergence, with initial conditions $r_{i,t}(0) = \frac{1}{|V|}$. $\vec{r}_t$ is computed for each timestamp, i.e., year, $t \in T$. In the following, we often refer to the PageRank vector as $\vec{r}$ neglecting the subscript.

In Figure 6.6 we graphically show the PageRank in 2014, while Table 6.3 reports the rank of the 20 countries having highest PageRank in 2000, 2014, and 2016. As stated above, the drain index does not privilege nodes having high both in-strength and out-strength, and does not account for the importance of the origin/destination of the connections. PageRank is instead able to picture such aspects. For example, all the countries highlighted in bold in Table 6.1 are among the best 10

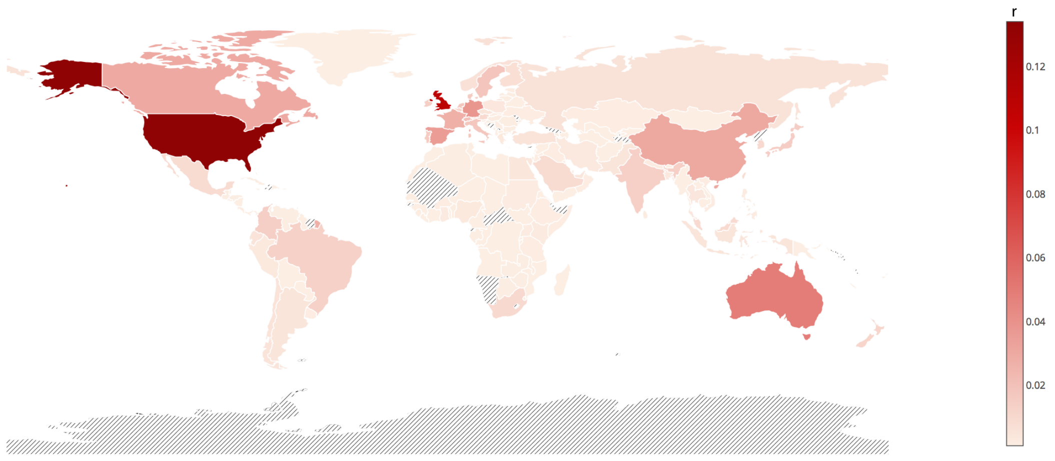


Figure 6.6: PageRank r in 2014. Countries without data are depicted in white with diagonal lines.

Table 6.3: Top-20 ranking by PageRank in 2000, 2014, and 2016.

ranking	2000	2014	2016
1	United States	United States	United States
2	United Kingdom	United Kingdom	United Kingdom
3	Germany	Australia	Australia
4	Spain	Spain	Germany
5	Italy	Germany	Spain
6	France	China	China
7	Canada	France	Canada
8	Australia	Canada	France
9	Portugal	Italy	Switzerland
10	Netherlands	Sweden	Sweden
11	Sweden	Portugal	Netherlands
12	Japan	Brazil	Italy
13	Switzerland	Switzerland	Denmark
14	Brazil	Netherlands	Portugal
15	China	Denmark	Japan
16	South Korea	India	Ireland
17	Malaysia	Japan	Colombia
18	Mexico	South Korea	India
19	Denmark	Belgium	Brazil
20	Indonesia	Saudi Arabia	New Zealand

countries in terms of PageRank in 2014; in particular, United States and United Kingdom place at the first and at the second position of the ranking, respectively.

On the whole, PageRank is confirmed to be a powerful method to rank the nodes of a network. However, it assigns to each node a unique score and it is not desirable in our setting, since we are instead interested in understand the interplay between importing and exporting researchers. Therefore, our analysis is required to rely on more refined and specific metrics that highlight such duality.

6.2.3 A more refined approach: hubs and authorities

We identify the *hyperlink-induced topic search* algorithm (also known as HITS or *hubs and authorities*) [143] as the ultimate tool to study our network. The HITS hub vector $\vec{h}_t = (h_{1,t}, \dots, h_{|V|,t})^\intercal$ and the HITS authority vector $\vec{a}_t = (a_{1,t}, \dots, a_{|V|,t})^\intercal$ in $t \in T$ of $G = (V, T, \varpi)$ are defined by the

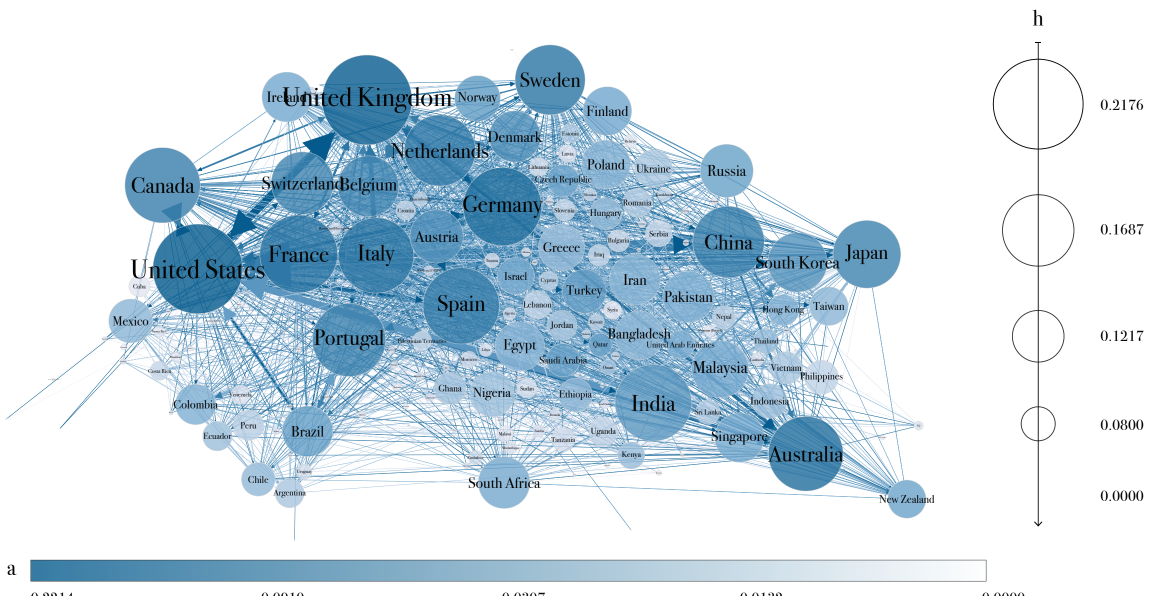


Figure 6.7: Geographic layout of the scientific migration network in 2014. The dimension of a node $i \in V$ represents h_i, while the color represents a_i. Edge thickness stands for edge weight.

limit of the following set of iterations:

$$\vec{h}_t(x+1) = c_t(x)W_t\vec{a}_t(x+1) \tag{6.4}$$

and

$$\vec{a}_t(x+1) = d_t(x)W_t^{\intercal}\vec{h}_t(x), \tag{6.5}$$

where $c_t(x)$ and $d_t(x)$ are normalization factors to make the sums of all elements become unity, i.e., $\sum_{i=1}^{|V|} h_{i,t}(x+1) = 1$ and $\sum_{i=1}^{|V|} a_{i,t}(x+1) = 1$. The initial HITS values of the scores are $h_{i,t}(0) = \frac{1}{|V|}$ and $a_{i,t}(0) = \frac{1}{|V|}$ for all $i \in V$. Note that, in this work, we employ the weighted version of HITS. The non-weighted HITS hub scores and non-weighted HITS authority scores are defined in the exactly the same way, replacing W_t with the unweighted adjacency matrix in Equations 6.5 and 6.4. Also in this case, $\vec{h}_t$ and $\vec{a}_t$ are computed for each timestamp, i.e., year, $t \in T$. In the following, we often refer to the HITS hub and authority vectors as $\vec{h}$ and $\vec{a}$ neglecting the subscript.

By definitions, a node $i \in V$ has large value of a_i if it has many links of large weight towards those nodes $j \in V$ having high h_i; similarly, node i has large value of h_i if it is reached by nodes $j \in V$ of high a_i throughout links of large weight. In our specific scenario, $\vec{a}$ provides an indication of which are the *provider* countries, that export many researchers in direction of the most attractive countries; while $\vec{h}$ indicates which are the *attractor* countries, able to attract many researchers from important exporters. Figure 6.7 shows hub and authority scores for the scientific migration network in 2014. Unites States and United Kingdom stand out from the plot: they place first and third in the hub ranking, and first and second in the authority ranking, respectively. In general, North American and European countries are represented by big circles and in dark color, since they have high values of both hub and authority scores; the same is for Australia. India, instead, has large dimension but milder color because it results to be among the top exporters but not as attractive as other countries.

Tables 6.4 and 6.5 show the first twenty countries ordered by hub score and authority score, respectively, in 2000, 2014, and 2016. China, United States, and United Kingdom are identified as the leading provider countries during the whole time domain: they never fall below the fifth position. India and Canada, followed by various of European countries, i.e., Germany, Italy, Spain, and France, consistently position after the three leading countries with few fluctuations during the

Table 6.4: Best providers of scientist: top-20 ranking by hub score in 2000, 2014, and 2016.

ranking	2000	2014	2016
1	China	China	United States
2	United Kingdom	United Kingdom	China
3	Canada	United States	United Kingdom
4	United States	India	Germany
5	South Korea	Spain	India
6	France	Canada	Spain
7	Germany	Italy	Canada
8	India	Germany	Italy
9	Italy	France	Australia
10	Spain	Brazil	France
11	Australia	Australia	Netherlands
12	Japan	Portugal	Brazil
13	Brazil	South Korea	Switzerland
14	Russia	Netherlands	Portugal
15	Portugal	Japan	South Korea
16	Mexico	Switzerland	Sweden
17	Turkey	Sweden	Japan
18	Switzerland	Iran	Denmark
19	Colombia	Turkey	Ireland
20	Taiwan	Colombia	Belgium

Table 6.5: Best attractors of scientist: top-20 ranking by authority score in 2000, 2014, and 2016.

ranking	2000	2014	2016
1	United States	United States	United States
2	United Kingdom	United Kingdom	United Kingdom
3	Germany	Australia	Australia
4	Italy	Germany	Germany
5	Spain	France	Canada
6	Canada	Canada	Spain
7	Australia	Spain	China
8	Portugal	China	France
9	France	Italy	Switzerland
10	Japan	Portugal	Netherlands
11	Netherlands	Sweden	Sweden
12	South Korea	Switzerland	Japan
13	Sweden	South Korea	Italy
14	Brazil	Netherlands	Denmark
15	Malaysia	Brazil	Portugal
16	Switzerland	Denmark	Hong Kong
17	China	Japan	Ireland
18	Ireland	Hong Kong	Colombia
19	Mexico	India	Singapore
20	Taiwan	Singapore	India

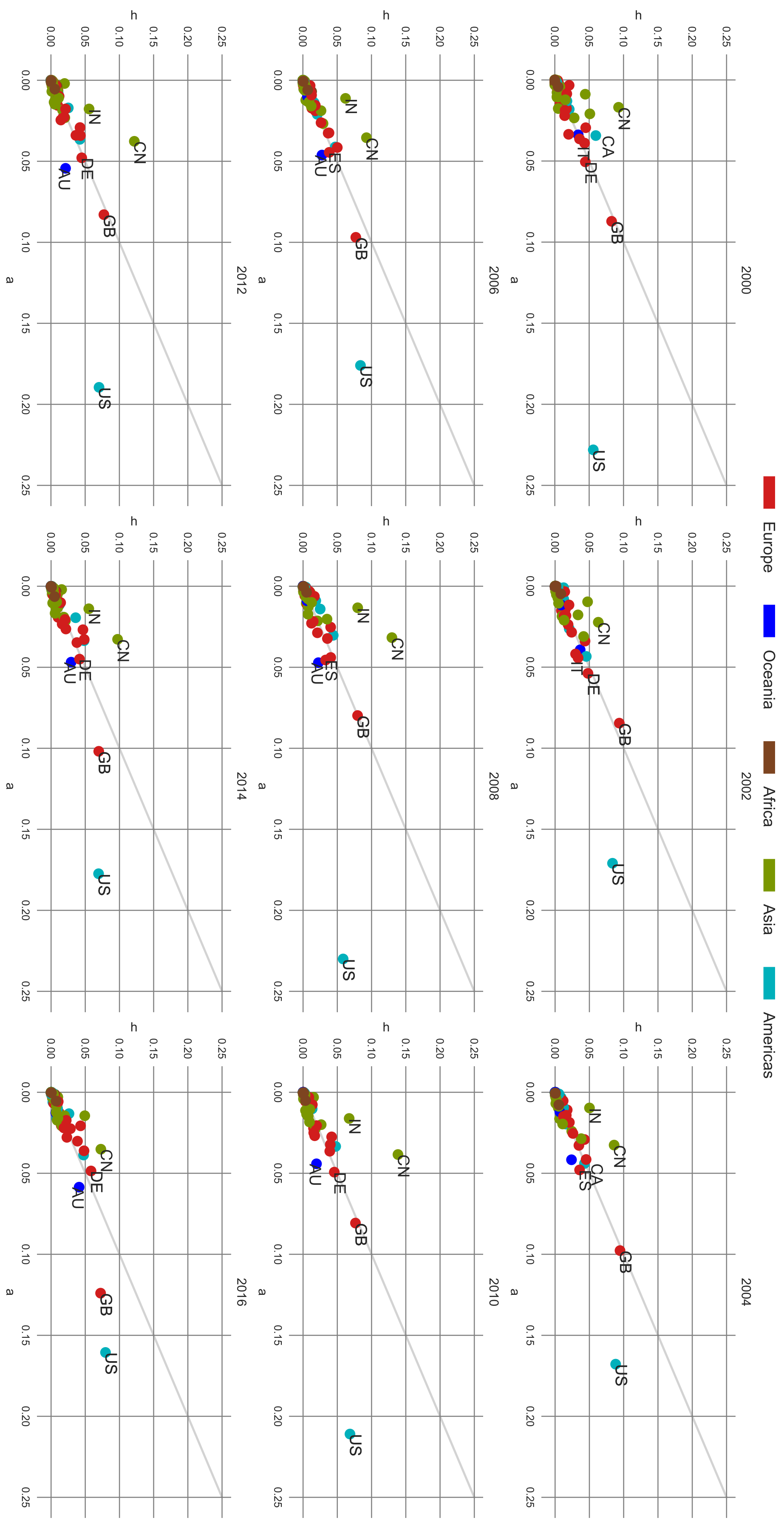


Figure 6.8: Evolution of hub and authority scores of the nodes of the scientific migration network in time. ISO 3166-1 alpha-2 codes are reported for selected countries: Australia (AU), China (CN), Germany (GE), India (IN), Italy (IT), Spain (ES), United Kingdom (GB), and United States (US).

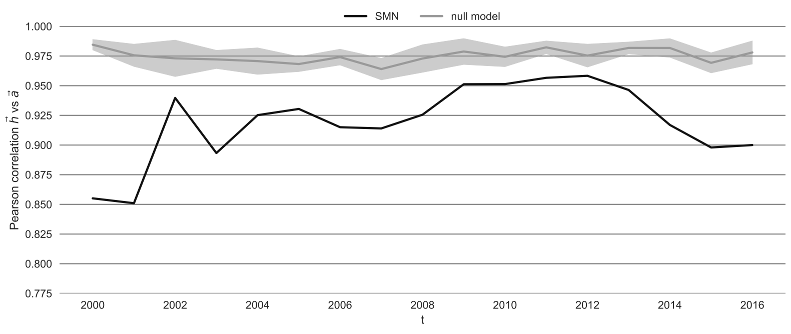


Figure 6.9: Person correlation between $\vec{h}$ and $\vec{a}$ of the scientific migration network and of the null model, for which we report mean and 95% confidence interval. p-values are smaller than $1.5e{-}05$ in all cases.

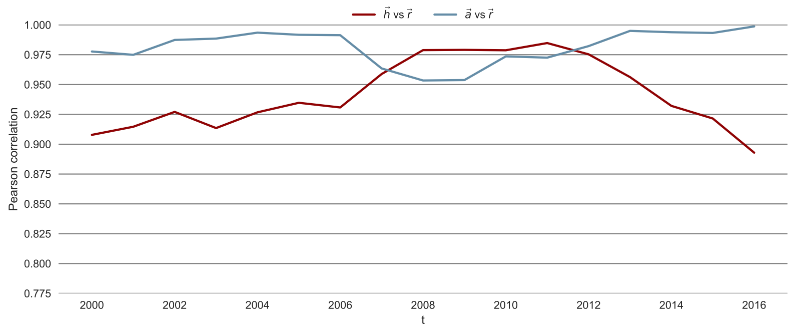


Figure 6.10: Person correlation between $\vec{h}$ and $\vec{a}$, and $\vec{r}$ of the scientific migration network.

years. South Korea and Russia follow instead negative trends. South Korea is the fifth hub in the scientific migration network during 2000, then loses ten positions by 2016. Russia's decay is even worse: it is among the best twenty hubs in 2000, leaves the top-20 in 2003, and touches the 38th position in 2016. About the authority score, United States have the best performance during the whole time horizon, while United Kingdom always classifies 2nd. Germany generally occupies the 3rd position in early 2000, before the growth of Australia. Similarly to the hub score, after the top-4 positions, there is a series of countries composed by the European Spain, France, and Italy, together with Canada and China. Interestingly, among the best attractors, there are Asiatic countries that are not identified as good hubs, e.g., South Korea, Singapore, and Hong Kong.

Figure 6.8 depicts the evolution of hub and authority scores of the nodes of the scientific migration network in time, by means of scatterplots. Ideally, we can state that a country is more important as hub than as authority if it places above the diagonal, and viceversa. In all years, most of the countries clump in the lower-left corner, where both scores are close to 0. A few countries differentiate from the others instead. United States are always more central with respect to the authority score than to the hub score, even if they are among the leading hubs overall. On the other hand, United Kingdom moves from being equally hub and authority in early '00 to being more authority by the end of the time domain. It is also easy to notice how China, which is constantly among the top hubs, slowly increases its authority score.

In light of this, the correlation between $\vec{h}$ and $\vec{a}$ and the evolution of such correlation is an interesting aspect to take into account. We show, in Figure 6.9, the Pearson correlation between $\vec{h}$ and $\vec{a}$ as a function of the year, and compare it to a null model. As null model we employ the *configuration model* [184] that allows to test whether the correlation is a non-trivial feature of the scientific migration network or it is expected by the strength distribution of the nodes. The configuration model rewires the edges preserving the strength distribution of the nodes in each year, namely, an edge can be shuffled only with other edges with the same timestamp. Note that by this hypothesis, in the resulting null model, the edge weight distribution and the number of

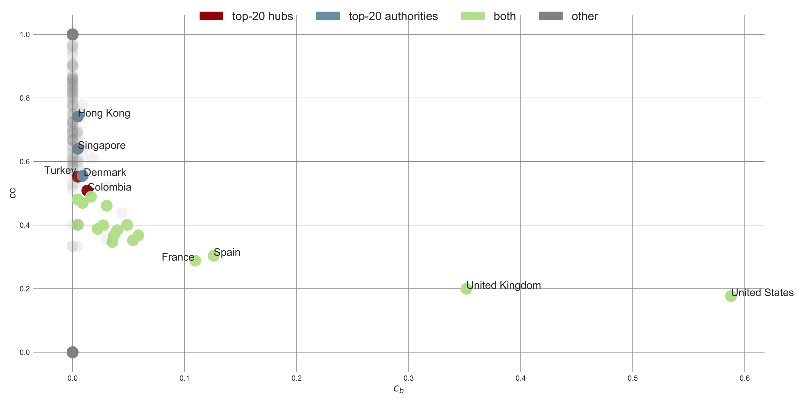


Figure 6.11: Betweenness centrality (x-axis) against clustering coefficient (y-axes) of the nodes of the scientific migration network in 2014.

edges in each year might vary with respect to the original network. Here and in the following results, we consider ten different configurations of the null model. The correlation in the original network is strong during the whole time domain, constantly greater than 0.85. The null model has even stronger correlation in all years, with small variation between the different configurations. This means that we should expect more countries of high (low) hub score having also high (low) authority score, and viceversa, in the scientific migration network. The observed behavior should then rely on different factors, e.g., local patterns – which we study in the next section – than the strength distribution.

In order to compare the HITS and the PageRank results, in Figure 6.10 we also visualize the Pearson correlation between $\vec{h}$ and $\vec{a}$, and $\vec{r}$. Interestingly, both $\vec{h}$ and $\vec{a}$ are highly correlated to $\vec{r}$. $\vec{a}$, in particular, has correlation greater than 0.95 in all years. This validates the results obtained by the HITS algorithm that has the advantage of depicting two different aspects of the world countries, providing then more accurate indications.

6.2.4 Betweenness centrality vs clustering coefficient

Besides the role that a country have in the overall scientific migration network, it is of our interest to understand how the countries position and influence their local neighborhood and community.

We define the *betweenness centrality* of a node $i \in V$ at time $t \in T$ as

$$c_b(i,t) = \sum_{\substack{s,e \in V \\ i \neq s \neq e}} \frac{\sigma_{se,t}(i)}{\sigma_{se,t}}, \tag{6.6}$$

where $\sigma_{se,t}$ is the total number of shortest paths from node s to node e at time t, and $\sigma_{se,t}(i)$ is the number of such paths passing through node i. In the computation of the betweenness centrality, we consider the reciprocal of the edge weights of the scientific migration network, since the more a path is favorable (i.e., shorter) the more researchers move through such path. Therefore, c_b is an indication of how much a country is central in the crossing of the network by the researchers. Usually, countries of high betweenness centrality place at the borders of their local clusters and have direct ties towards other clusters. Therefore, we can suppose that such countries are one of the two endpoints of a *bridge*, or more likely of a *local bridge* [73] (local bridge is a relaxed definition of bridge, i.e., if we delete a local bridge the two endpoints would lie further away and not in two different components of the network). The endpoints of a (local) bridge regulate the access toward different clusters of nodes and are crossroads of the flows within the network. Hence, countries like the United States and the United Kingdom are important players in the scientific migration network since the scientific migration moves also ideas and information in addiction to people: these countries may have early access to knowledge and to new research results, possibly produced in multiple and non-interacting places of the world.

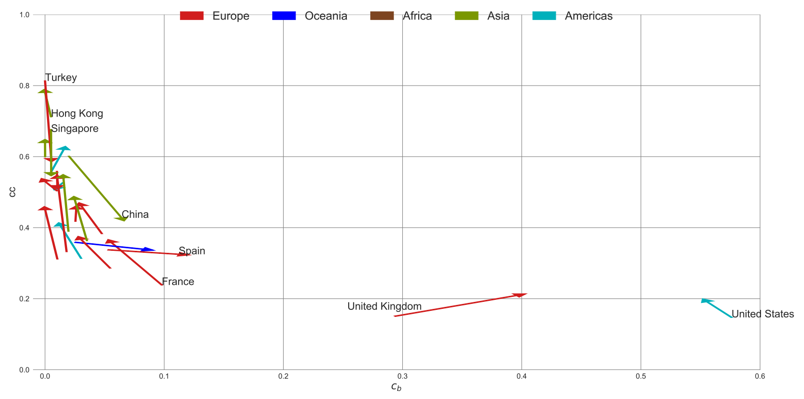


Figure 6.12: Betweenness centrality-clustering coefficient trajectories of selected countries (top-20 hubs and top-20 authorities in 2014) from 2000 to 2016.

We also compute the *clustering coefficient* of a node $i \in V$ at time $t \in T$ as

$$cc(i,t) = \frac{|(j,k) \mid j,k \in N_{i,t} \wedge (j,k) \in E_t|}{|N_{i,t}|(|N_{i,t}|-1)}, \tag{6.7}$$

where $N_{i,t}$ identifies the neighbor set of node i at time t. In this case, we neglect the edge weights. Also note that, this definition, differently than the definition of betweenness centrality, treats the G as undirected. In our context, we consider the cc of a country i as a measure of how many possible origins or destinations the researchers residing in neighbor countries have rather than i.

Figure 6.11 reports betweenness centrality (x-axis) against clustering coefficient (y-axes) of the nodes of the scientific migration network in 2014, highlighting the top-20 hubs and the top-20 authorities. Most of the countries place in the upper-left corner of the plot, having high clustering coefficient (i.e., the neighbor countries have many other connections between them) and low betweenness centrality (i.e., they are internal to their local clusters). Interestingly, none the highlighted countries (with the exception Hong Kong and Singapore) is in such position. Rather, the main hubs and authorities of the scientific migration network tend to be central in the migration paths traversing the network, and influence their local neighborhood centralizing the connections towards them. In particular, United States, United Kingdom, Spain, and France stand out from the others. Again, Hong Kong and Singapore are exceptions, having a behavior common to most of the countries.

As a further step in this direction, in Figure 6.12, we report the trajectories of the twenty countries of highest hub score and the twenty countries of highest authority score of year 2014 in terms of betweenness centrality and clustering coefficient over the time span under analysis. Each arrow of the plot is associated to a country: the root represents the country in 2000, while the head shows the same country in 2016. Despite our previous observations, we cannot observe a global pattern, common to most of the countries, leading toward the lower-right corner: some of the nodes move towards the upper-left corner, others to more favored positions. For most of the European countries, betweenness centrality decreases and clustering coefficient increases. This behavior, which is frequently observed when a set of nodes tighten its cluster structure, might reveal the adoption of the new migration polices provided by the rising European Union, during nineties and noughties. Spain and United Kingdom are the most evident exceptions, probably because they played a key role in bridging toward the Spanish-speaking countries of Latin America and the former Commonwealth states, respectively. Moreover, all countries move around their surroundings. China has the greater improvement combining betweenness centrality and clustering coefficient, while Turkey has the highest variation in terms of clustering coefficient. Note that, by considering the reciprocal of the edge weights in the computation of the betweenness centrality, a country is required to either polarize the distribution of its weights or increase its strength to augment such centrality. In the next section, we rely on the study of statistical dispersion of incoming and outgoing edge weights to provide a better understanding of such local patterns.

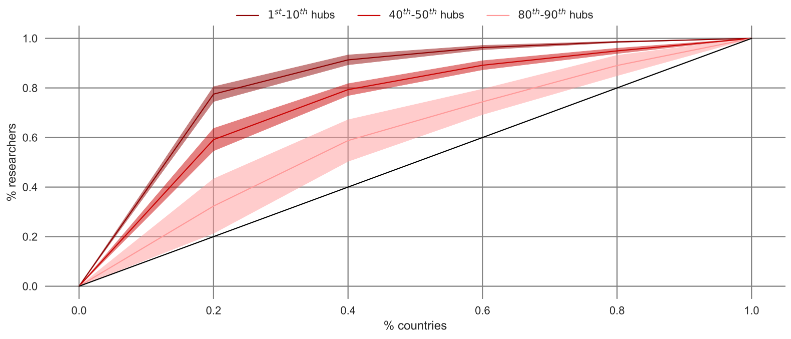


Figure 6.13: Lorenz curves and 95% confidence intervals for three classes of hubs in 2014. The population **W** is represented by the edge weights of incoming edges.

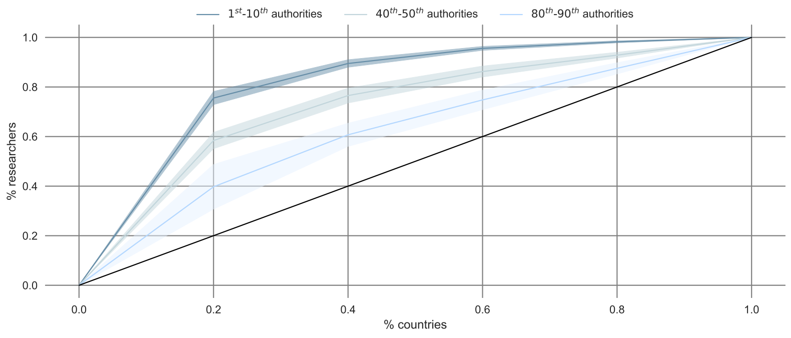


Figure 6.14: Lorenz curves and 95% confidence intervals for three classes of authorities in 2014. The population **W** is represented by the edge weights of outgoing edges.

6.3 Local patterns

In this section, we dive deeper into the factors that contribute to establish a country as leading hub or authority in the scientific migration network.

6.3.1 Predecessors and successors

At first, we investigate the homogeneity of the edge weights of the neighborhood of the nodes. Specifically, we want to understand how the researchers leaving (reaching) a country of high hub (authority) score distributes with respect to the predecessors (successors) of such country. In order to do so, we employ the *Gini coefficient*, which measures the degree of inequality of a distribution [111]. Given a population $\mathbf{W} = \{w_o, w_1, \ldots, w_n\}$ of n values, we define the Gini coefficient as

$$G = \frac{\sum_{w_i, w_j \in \mathbf{W}} |w_i - w_j|}{2n \sum_{w_i \in \mathbf{W}} w_i}. \tag{6.8}$$

G varies between 1 and 0, where 1 expresses maximal inequality among values while 0 indicates the case in which all the values in **W** are equal.

In the following, we graphically show the Gini coefficient by means of Lorenz curves identifying the population **W** as the edge weights of outgoing edges or the edge weights of incoming edges when considering a node as hub or authority, respectively. Therefore, we aim at investigating how (un)balanced the migration flows from/towards a country are and how such aspect correlates to $\vec{h}$ and $\vec{a}$. Figures 6.13 and 6.14 compare the mean Lorenz curves, along with 95% confidence intervals, of three different classes of hubs and authorities, respectively. It is immediate to notice that high hub/authority score is associated with high Gini coefficient. The Gini coefficient decreases progressively as we move down with the hub and authority rankings. Then, to obtain a leading

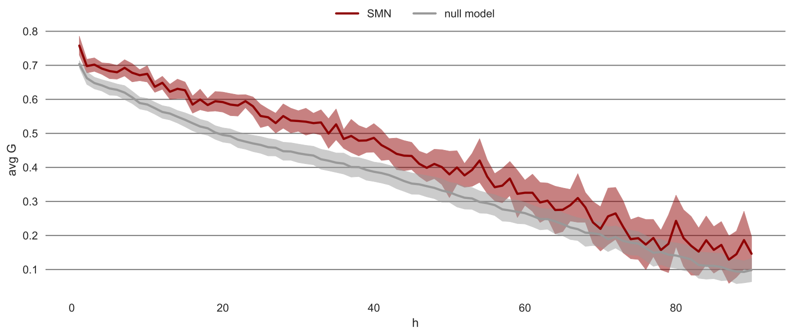


Figure 6.15: Average Gini coefficient (and 95% confidence interval) as a function of the hub ranking of the scientific migration network and of the null model. The population **W** is represented by the edge weights of outgoing edges and the average is computed over the time domain T.

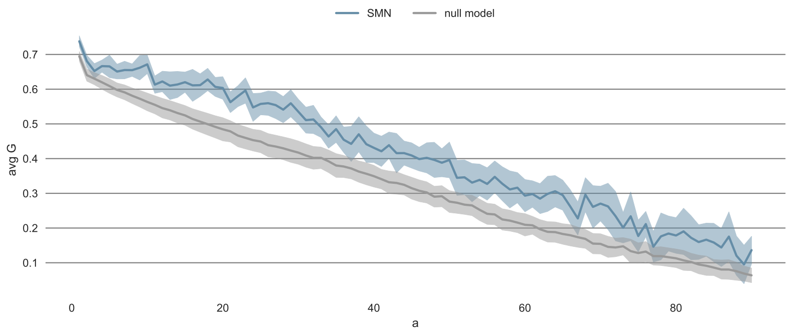


Figure 6.16: Average Gini coefficient (and 95% confidence interval) as a function of the authority ranking of the scientific migration network and of the null model. The population **W** is represented by the edge weights of outgoing edges and the average is computed over the time domain T.

position in the scientific migration network, a country is required to have strongly differentiated migratory flows from/towards its neighbors.

The behavior of the missing classes is consistent, as shown in Figures 6.15 and 6.16 which report the average (over the time domain T) of the Gini coefficient (and the 95% confidence interval) as a function of the hub/authority ranking. Such curves are compared with the null model considering the average of the ten different configurations we generate. The Gini coefficient decreases as h and a drop, both in the scientific migration network and in the null model, and the curves have very similar functional shapes. The confidence intervals are quite limited in all cases, however they become larger for the lowest positions of the ranking in the scientific migration network where data become more sparse and less significant. The Gini coefficient of the scientific migration network is (slightly) higher than the null model, then a node to be in the first positions of the hub/authority ranking is required to have high disparity in the weights of the connections from/to its predecessors/successors by the intrinsic characteristics of the network. Therefore, for a country, having preferential massive exchanges of researchers with partner states is more profitable than having a bunch of similar relationships and fundamental to stand out in the scientific migration phenomenon.

6.3.2 Clustering coefficient

Similarly to the Gini index, we study the behavior of the clustering coefficient (introduced in Equation 6.7) of the successors of the hubs and of the predecessors of the authorities as a function of the hub/authority ranking in the scientific migration network compared to the null model. These results vary from the ones presented in Section 6.2 since, in that case, we calculate the clustering coefficient over the whole neighborhood of a node, while here we are interested only in the subset

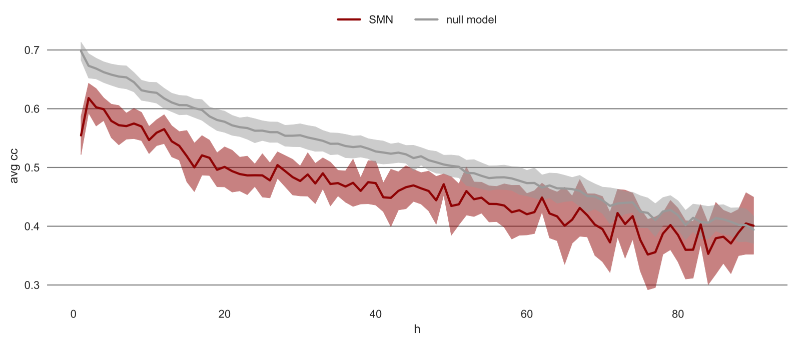


Figure 6.17: Average clustering coefficient (and 95% confidence interval) as a function of the hub ranking of the scientific migration network and of the null model. The average is computed over the time domain T.

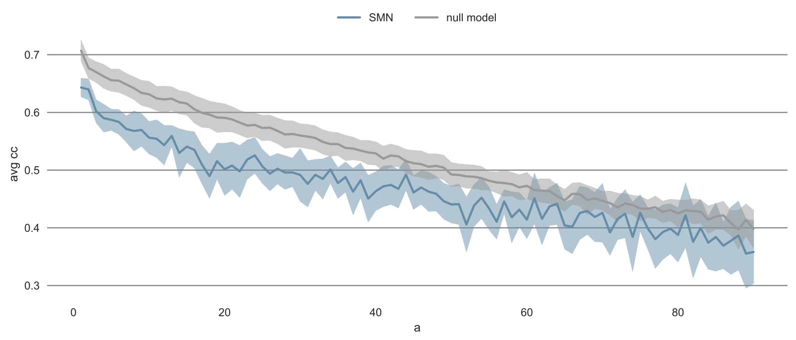


Figure 6.18: Average clustering coefficient (and 95% confidence interval) as a function of the authority ranking of the scientific migration network and of the null model. The average is computed over the time domain T.

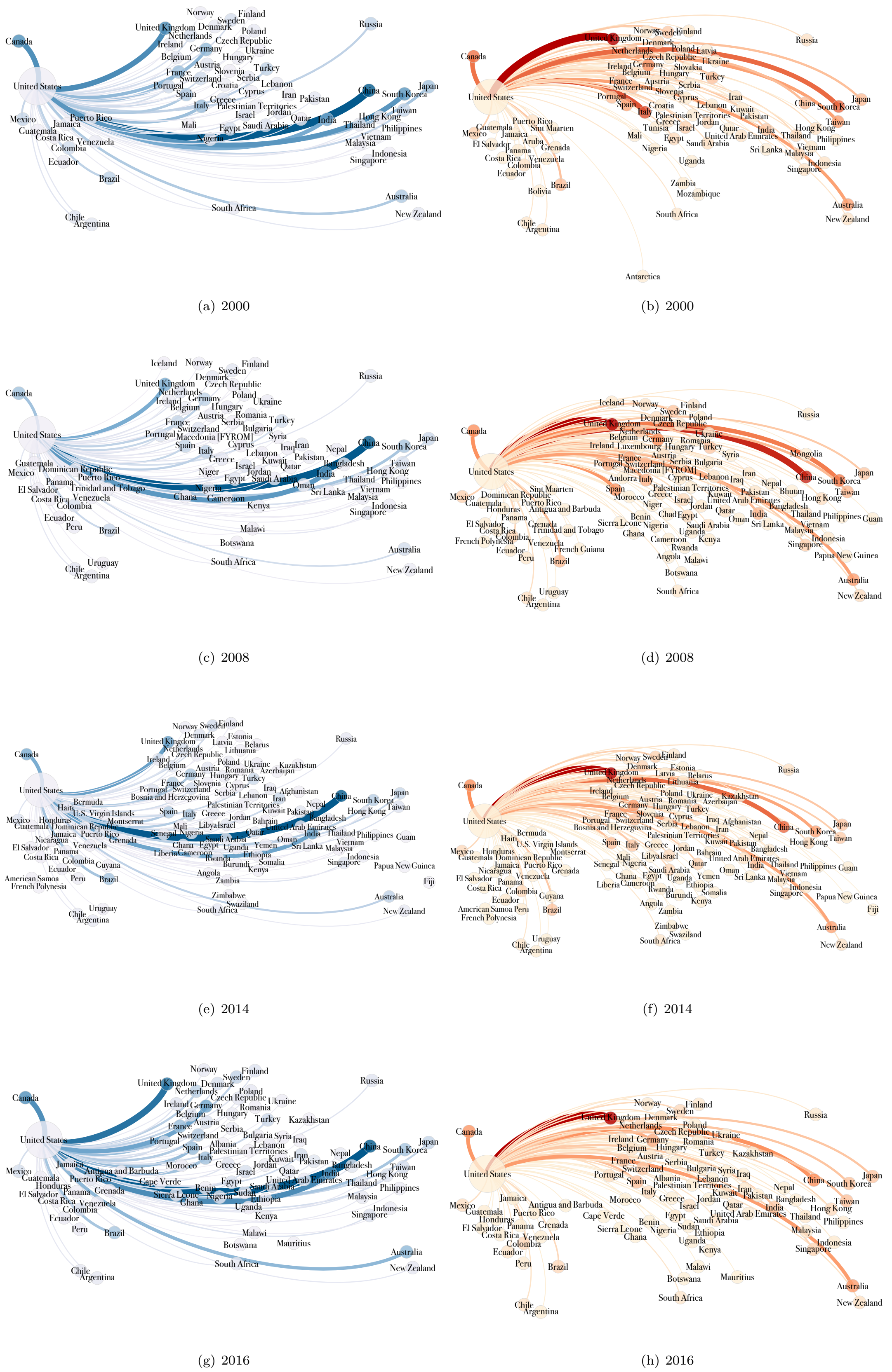
of the neighbors that counts in the computation of the hub/authority score.

In Figures 6.17 and 6.18 we observe that nodes of better ranking have higher clustering coefficient, both hubs and authorities. This observation reflects the fact that the higher the hub/authority score is the more the successors/predecessors of a country are cohesive, i.e., they are in the most active parts of the network. Also in this case, the trend observed in the null model is similar to the scientific migration network; however, the clustering coefficient of the null model is constantly greater with respect to the real network, in particular for the top hubs and authorities. Therefore, the best hubs/authorities of the scientific migration network are able to significantly influence (with respect to the null model) the local cluster structure, attracting most of the migratory connections towards them and breaking connections between neighbor countries.

6.4 Case studies

In this section we show how to exploit network visualization to evince temporal evolution of (partial) ego-networks of select hubs and authorities. In particular, we focus our attention on the connections between the focal node (i.e., the ego) and its neighbors, omitting edges whose endpoints do not include the ego. Given the nature of our study, for each select country we define two different visualizations; the first one depicts incoming migratory flows only, while the other separately shows outgoing connections. We retain that such visualizations are able to provide clear indications of the evolution in time of the characteristics and of the connection of a country in the scientific migration network.

Figures 6.19, 6.20, and 6.21 visualize the ego-networks of the United States, China, and Spain in 2000, 2008, 2014, and 2016. Colors and thickness of the edges, both normalized according to each ego-network, refer to edge weights. Moreover, the states are placed on the basis of their



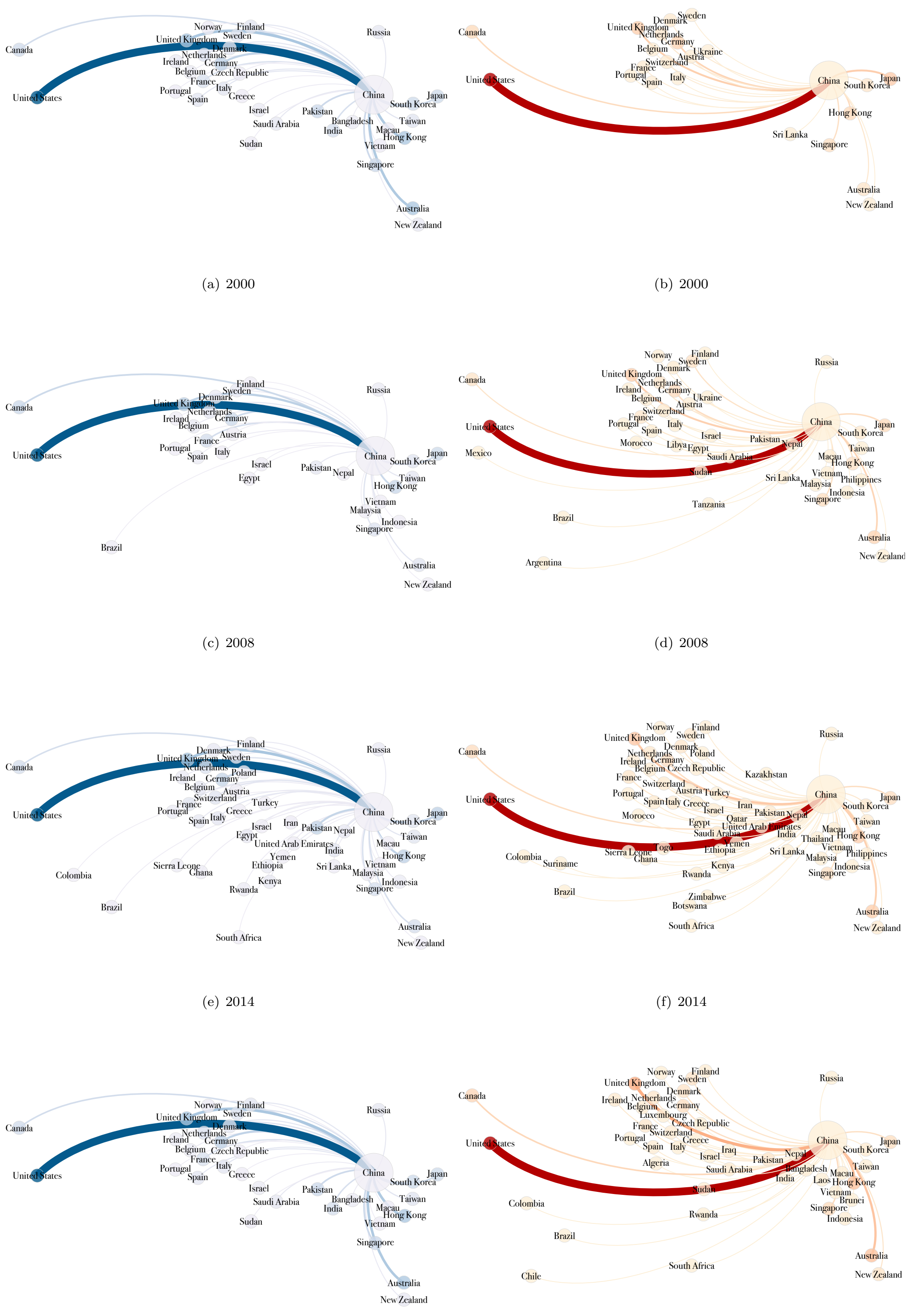
(a) 2000 (b) 2000

(c) 2008 (d) 2008

(e) 2014 (f) 2014

(g) 2016 (h) 2016

Figure 6.19: Ego-network evolution of United States: incoming connections (left, blue) and outgoing connections (right, red).



(a) 2000 (b) 2000

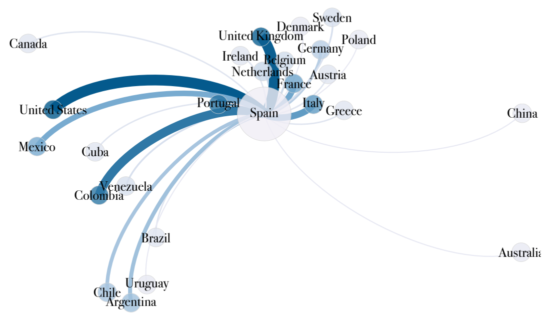
(c) 2008 (d) 2008

(e) 2014 (f) 2014

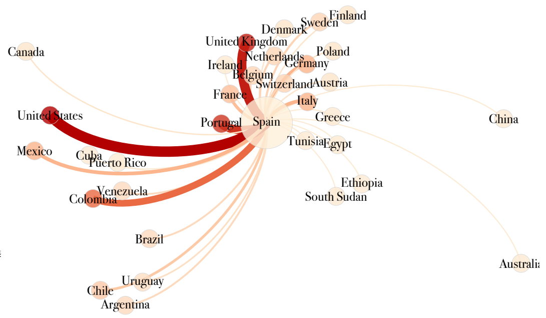
(g) 2016 (h) 2016

Figure 6.20: Ego-network evolution of China: incoming connections (left, blue) and outgoing connections (right, red).

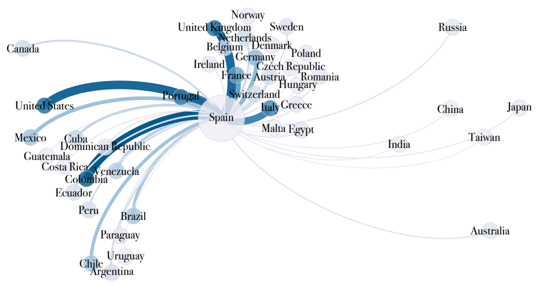
]



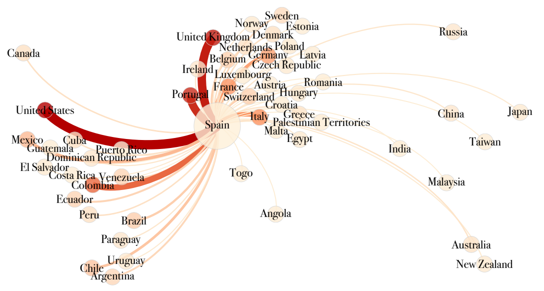
(a) 2000



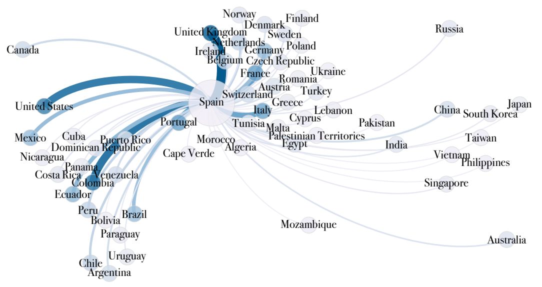
(b) 2000



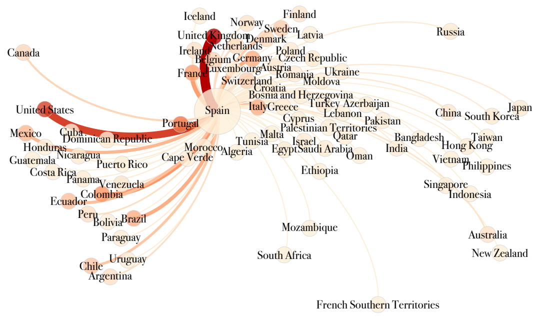
(c) 2008



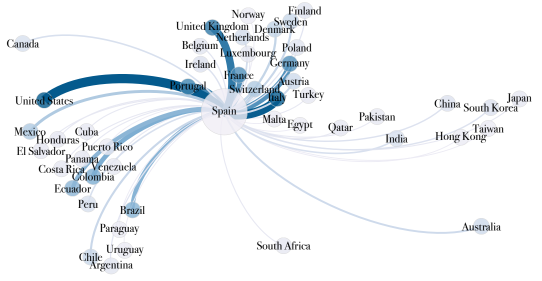
(d) 2008



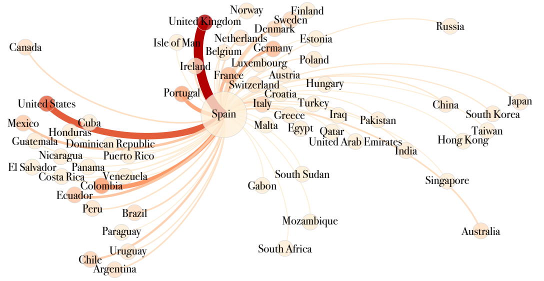
(e) 2014



(f) 2014



(g) 2016



(h) 2016

Figure 6.21: Ego-network evolution of Spain: incoming connections (left, blue) and outgoing connections (right, red).

geographical location.

It is easy to see (Figure 6.19) how the United States have many both incoming and outgoing migration channels. Their ego-networks are dense even in 2000, where the scientific migration network is sparser than in later years. The United States have constantly the best authority score but they occupy a very competitive position in the hub ranking too, which justifies the structure of their connections. In 2016, the United States and China have very close hub score (1st and 2nd in the hub ranking, respectively) but different authority score (1st and 7th in the authority ranking, respectively). In fact, looking at Figures 6.19(g), 6.19(h), 6.20(g), and 6.20(h), we notice two different migration models. The United States have many neighbor countries spread across all the continents. On the other hand, China is the major provider of researcher of the United States and the majority of outgoing researchers from China move to the US. On the contrary, China is only one of the many possible destinations for American researchers. Finally, Figure 6.21 highlights that Spain, whose trajectory in Figure 6.12 is common to local bridges, retains favored relationships with the Spanish-speaking countries of the Latin America.

6.5 Summary

In this chapter we study international migrations of the scientific population from a complex-network perspective, and we describe measures and patterns to identify the central countries involved in the migration phenomenon. In particular, we employ the HITS algorithm with the intent of catching the interplay between exporting and importing researchers from a global perspective. We also investigate the local characteristics of successors of hubs and predecessors of authorities to dive deeper into the motivations that establish hubs and authorities. Interestingly, our findings identify a set of countries that occupies a privileged position in the scientific migration network, being both important hubs and central authorities. The majority of such countries shares similar local characteristics/patterns, namely they exchange with many different states instead of having a few well-established migration corridors. At the same time, the migration flows are very unbalanced, as testified by the Gini coefficient. China is the most notable exception, having a favorite relationship with the USA in terms of return rate of researchers. Moreover, we observe different behaviours that lead actors with similar hub or authority score to occupy different positions in the community structure of the scientific migration network, preferring, e.g., to cooperate, as most of the European nations, rather than to act independently, such as China and United Kingdom. Such network dynamics deserves to be further analyzed for undercovering latent causes and factors by the inclusion of complementary sources, e.g., local regulations, political alliances, investments in research, development, and education. It is important to remark that all these findings should not be considered conclusive results due to the incompleteness and biases affecting the data (as already pointed out in Section 6.1).

In this work we apply the proposed methodology to data extracted from the ORCID platform. However, it is important to mention that our model is completely data-agnostic, meaning that it can be applied to other datasets obtained from different sources with no modifications. Moreover, it is able to accommodate evolving datasets that grows over time, delivering a more precise picture as the information increases.

Chapter 7

Span-cores in face-to-face interaction networks

Nowadays distributed sensing systems and infrastructures are based on wearable sensors that allow the gathering of data about proximity relations and close-range interactions of individuals in real-world large-scale settings [133, 203]. Different works [203, 217, 174, 204] collected data on the time-resolved face-to-face proximity of students and teachers in schools with the aim of studying mixing patterns of children in school environments. Such patterns would help to quantify the transmission opportunities of respiratory infections and to identify situations within schools where the risk of transmission is higher [176, 113].

In this chapter we use the three face-to-face interaction networks gathered in schools introduced in Chapter 4, i.e., PrimarySchool, HighSchool, and HongKong (see the beginning of Section 4.5 for the details), to illustrate applications of (maximal) span-cores and temporal community search in real-life analyses. The window size of all datasets is 5 minutes and, in the analysis, we discard span-cores of $|\Delta| = 1$, i.e., having span of 5 minutes, since they represent short interactions, not significant for our purposes. In the remaining of this chapter we describe (*i*) three types of interesting temporal patterns (Section 7.1), i.e., social activities of groups of students within a school day, mixing of gender and class, and length of social interactions in groups; (*ii*) a procedure to detect anomalous contacts and intervals that exploits maximal span-cores (Section 7.2); and, (*iii*) an approach to graph classification based on temporal community search (Section 7.3).

Notations, definitions, and algorithms employed here are directly borrowed from Chapter 4.

7.1 Temporal patterns

7.1.1 Temporal activity

We first show how span-cores (Definition 4.1) afford a simple temporal analysis of social activities of groups of people within a school day. The left side of Figure 7.1 reports colormaps of the order k of the span-cores as a function of their starting time t_s (x-axis) and of the size $|\Delta|$ of their temporal span $\Delta = [t_s, t_e]$ (y-axis), for a school day of the PrimarySchool and HighSchool datasets. Darker gray indicates span-cores of high order and slots located in the upper part of the plots refer to span-cores of long span. It is important to notice that the linear decay in span duration is naturally due to the definition of span-core and to the shifting of the starting time t_s; therefore, it is not a distinguishing feature of the activity patterns found in the analyzed data. In both datasets, fluctuations of k and $|\Delta|$ are observed along the day, which can be related to school events. Around 10 a.m., the size of the span $|\Delta|$ reaches a local maximum in correspondence to the morning break, which means that students establish long-lasting interactions that hold beyond the break itself. Moreover, when classes gather for the lunch break, the order k reaches its maximum value since students tend to form larger and more cohesive groups.

In order to verify that these results are not trivially derived from the general temporal activity, as simply given by the number of interactions in each timestamp, we compare our findings to a null

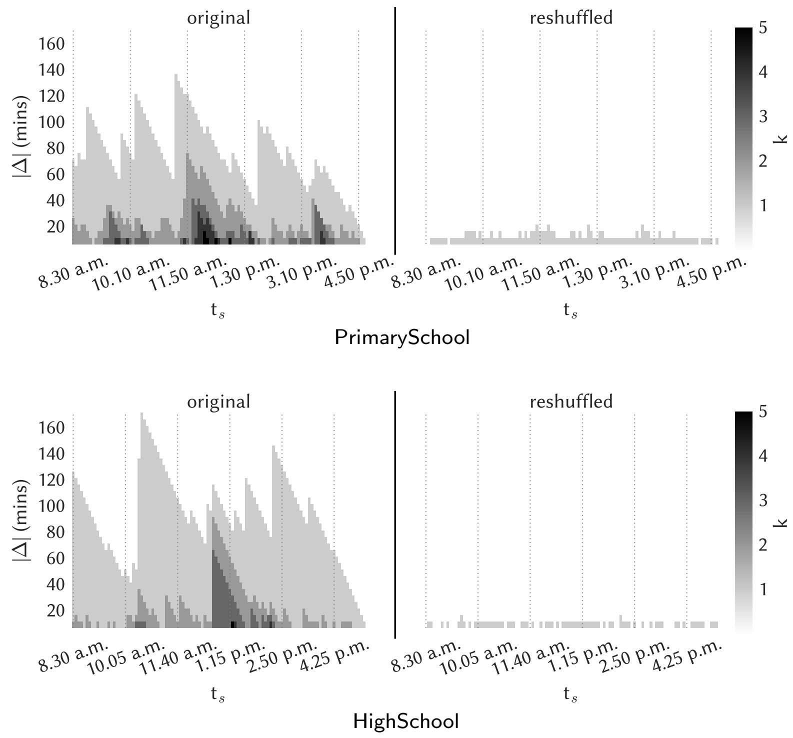


Figure 7.1: Temporal activity of a school day of the PrimarySchool and HighSchool datasets: the x-axis reports the hour of the day at which the span of a span-core starts, the y-axis specifies the size of the span (in minutes), and the color scale shows the order k. At a glance, it can be observed that the temporal structure of the span-core decomposition detects time-evolving cohesive structures in the original datasets (left plots) that completely disappear in the reshuffled datasets (right plots).

model. At each timestamp of the temporal graphs, we reshuffle the edges by the Maslov-Sneppen algorithm [173] which consists in repeating the following operations up to when all edges have been processed: select at random two edges with no common vertices, e.g., (u, v) and (w, z), and transform them into (u, z) and (w, v), if neither (u, z) and (w, v) existed in the original timestamp. This reshuffling preserves the degree of each vertex in each timestamp and the global activity (i.e., the number of contacts per timestamp), but destroys correlations between edges of successive timestamps. In the right side of Figure 7.1 we show the results of the temporal analysis described above for the reshuffled datasets. In both, the values of $|\Delta|$ and k reached are much smaller than in the original datasets. The size of the span $|\Delta|$ is always shorter than 20 minutes, while in the original datasets it is much longer, up to 170 minutes, and the order k is always equal to 1, compared to the original maximum of 5. The time-evolving cohesive structures detected by the temporal core decomposition in the original datasets are completely lost on reshuffling, since only span-cores of short span and low coreness are observed in the latter case. This shows that the temporal structure exposed by the span-core decomposition is not simply a consequence of temporal patterns of global activity but that span-cores represent a concrete method to detect complex cohesive structures and their temporal evolution.

7.1.2 Mixing patterns

We now show an analysis of mixing patterns of students with respect to gender and class. Such vertex attributes are indeed available for the individuals of the PrimarySchool dataset. We define as *gender purity* of a span-core the fraction of individuals of the most represented gender within the span-core. *Class purity* is analogously defined. The left plot of Figure 7.2 reports the temporal evolution of the average gender and class purity of the maximal span-cores (Definition 4.2) spanning

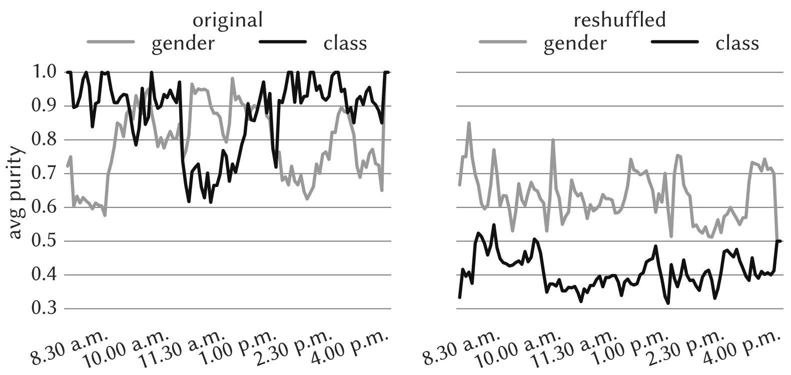


Figure 7.2: Temporal evolution (time on the x-axis) of average gender purity and average class purity (y-axis) of the maximal span-cores of the PrimarySchool dataset. Original data on the left, reshuffled data on the right.

each timestamp, during the first school day of the PrimarySchool dataset. During lessons, when students are in their own classes, class purity has naturally very high values, very close to 1. Gender purity is instead rather low. On the other hand, when students are gathered together, during the morning break at 10 a.m. and the lunch break between 12 a.m. and 2 p.m., the situation is overturned: gender purity reaches large values while class purity drastically decreases. This shows that primary school students group with individuals of the same class, disregarding the gender, only when they are forced by the schedule of the lessons, but prefer on average to form cohesive groups with students of the same gender during breaks. This is in agreement and complements a previous study of the same dataset focusing on single interactions in the static aggregated network [216].

The right plot of Figure 7.2 shows the temporal evolution of the average gender and class purity for a null model in which gender and class are randomly reshuffled among individuals. The two curves are more flat and the anti-correlation between them completely vanishes. This testifies that the results on the original dataset are not simply due to the relative abundance of individuals of each type interacting at each time, but reflect genuine mixing patterns and their temporal evolution.

7.1.3 Interaction length

Finally, we analyze the duration of interactions of social groups in schools by studying the distribution of the size of the span of the maximal span-cores of the three datasets (Figure 7.3). All distributions are extremely skewed with broad tails: most maximal span-cores have duration

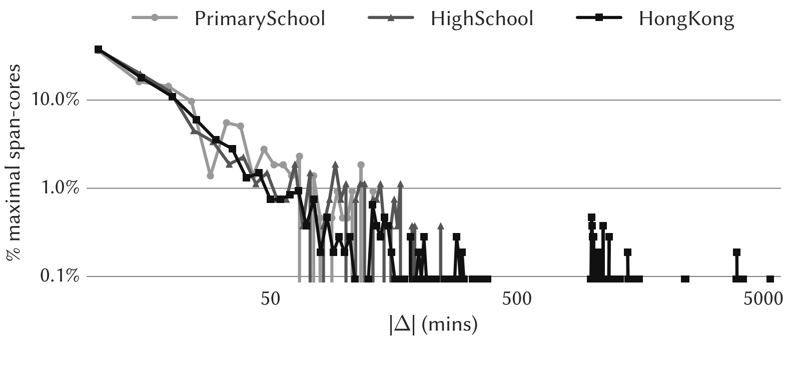


Figure 7.3: Distribution of the size of the span $|\Delta|$ of the maximal span-cores. The x-axis reports the size of the span (in minutes), while the y-axis the percentage of maximal span-cores having a given size of the span.

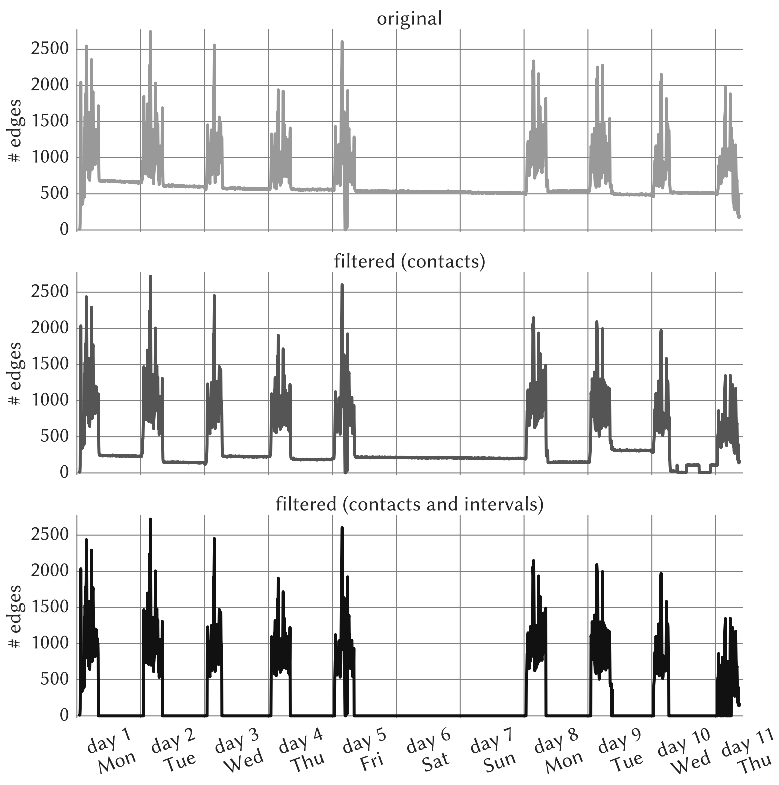


Figure 7.4: HongKong dataset: number of edges per timestamp in the original data (top), after filtering anomalous edges (middle), and after filtering anomalous edges and intervals (bottom). Days 6 and 7 are weekend.

less than 1 hour, but durations much larger than the average can also be observed. Interestingly, the three datasets at hand all exhibit the same functional shape, confirming a robust statistical behavior. We also note that similar robust broad distributions have been observed for simpler characteristics of human interactions such as the statistics of contact durations [217, 174]. Outliers appear also at very large durations, especially for the HongKong dataset that has maximal span-cores lasting up to 83 hours. Group interactions of such long span are clearly abnormal and represent outliers in the distributions. We will show, in the following of this section, how to exploit such outliers to detect both irregular interactions and anomalous temporal intervals.

7.2 Anomaly detection

The identification of anomalous behaviors in temporal networks has been the focus of several studies in the last few years [179, 204]. Based on the above findings, we devise a simple procedure to detect anomalous edges and intervals of the HongKong dataset that exploits maximal span-cores. The topmost plot of Figure 7.4 reports the number of edges for each timestamp of the original HongKong dataset. It is easy to notice that there is a lot of constant anomalous activity between school days and during the weekend, i.e., days six and seven: unexpectedly, the number of interactions per timestamp does not drop to zero. This happened in fact because proximity sensors were left in each class and close to each other, at the end of the lessons. In order to automatically detect these steady activity patterns that do not correspond to any genuine social dynamics, we apply the following procedure: (*i*) find a set of anomalously long temporal intervals supporting maximal span-cores, (*ii*) identify anomalous vertices, and, (*iii*) filter out anomalous edges.

The first step of this procedure requires to find the set of temporal intervals $\mathcal{I} = \{\Delta \sqsubseteq T \mid C_{k,\Delta} \in \mathbf{C}_M \wedge |\Delta| > tr\}$ that are the span of a maximal span-core $C_{k,\Delta}$ with size longer than a certain threshold tr. Then, for each timestamp $t \in T$, select as anomalous all those vertices that appear in the span-cores $\{C_{1,\Delta} \mid \Delta \in \mathcal{I} \wedge t \in \Delta\}$, i.e., the span-cores of $k = 1$ whose span is in $\mathcal{I}$ and contains t. Finally, at each timestamp $t \in T$, remove edges that are incident to at least a vertex that has been marked as anomalous at time t. Consistently with the distribution of the span durations of the maximal span-cores, we select the threshold $tr = 22$ (110 minutes). The results of this filtering procedure are shown in the middle plot of Figure 7.4. The number of edges during school days remains approximately unchanged, while the activity noticeably decreases in-between. Identifying as positives the spurious interactions occurring when the school is closed and as negatives the genuine interactions observed when the school is open, this approach achieves a precision of 0.91 and a recall of 0.64.

We can refine this anomaly detection process by identifying, in addition to anomalous edges, also anomalous temporal intervals. We define a timestamp $t \in T$ as anomalous if the ratio between the number of original edges (top plot of Figure 7.4) and the number of filtered edges (middle plot of Figure 7.4) exceeds a given threshold. We apply this further filtering to the HongKong dataset with a threshold of 1.5 and report the results in the bottommost plot of Figure 7.4. The number of edges when the school is closed drops to zero, while the activity during school days is not modified, except for the last one, which is affected by the proximity to the end of the time domain. The overall procedure yields a slightly higher value of precision, 0.93, and substantially improves the recall to 0.99.

7.3 Graph embedding and (supervised) vertex classification

In this section we show how Temporal Community Search (Problem 4.3) can be profitably exploited for classifying the vertices of a temporal graph. Specifically, the classification framework we set up is based on the paradigm of *graph embedding*, which has attracted a great deal of attention in the last few years, and whose goal is to assign to every vertex of a graph a numerical vector (i.e., an *embedding*) such that structurally similar vertices are represented by similar vectors, and vice versa [118, 77, 115]. Here, our framework simply consists in learning suitable embeddings for the vertices of the input graph, and then give them as input to some (well-established) classifier to ultimately accomplish the desired classification task. Thus, the main goal is to learn embeddings that are well-representative of the relationships among vertices, so as to help the classifier perform accurately. As our main result here, we show how an embedding strategy based on a simple exploitation of the output of Temporal Community Search achieves results comparable to well-established vertex-embedding methods such as DeepWalk [189], LINE [224], and node2vec [118].

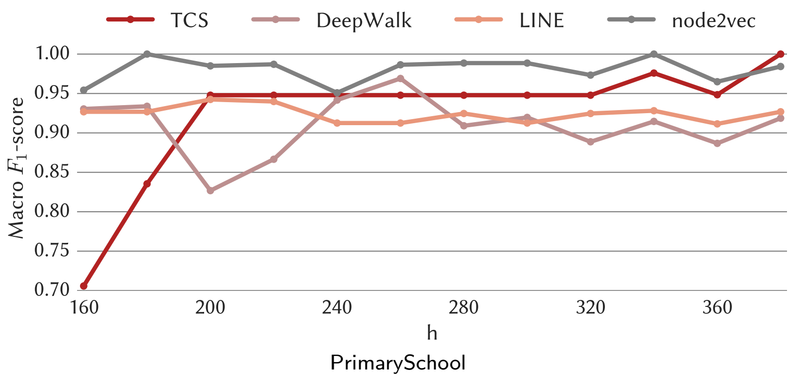
Method. For every vertex of the input temporal graph, we build an embedding as an h-dimensional vector conveying the information provided by a solution to the Temporal Community Search problem on the same graph. Specifically, consider a vertex $u \in V$ and a solution $\{\langle S_i, \Delta_i\rangle\}_{i=1}^{h}$ to Temporal Community Search on query-vertex set $Q = \{u\}$. We define u's embedding as

$$\mathbf{X}_u = [v^*_{Q,\Delta_1}, v^*_{Q,\Delta_2}, \dots, v^*_{Q,\Delta_h}], \tag{7.1}$$

which corresponds to the temporally-ordered sequence of minimum degrees of the h communities identified by the temporal-community-search solution. Below we show that this simple approach is sufficient to achieve interesting experimental results. Clearly, more sophisticated methods are possible, e.g., by simultaneously exploiting information from the S_i communities. However, our main goal here is to give an idea of how the Temporal Community Search problem can be successfully leveraged in a relevant application scenario, rather than devise the best temporal-community-search-based graph-embedding method.

Evaluation. We assess the performance of our method on the PrimarySchool and HighSchool datasets. In these datasets vertices correspond to students, and vertex labels (to be predicted) are the classes that every student belongs to. We involve in the comparison the following state-of-the-art vertex-embedding methods:

- DeepWalk [189], a method that preserves the proximity between vertices by running a set of random walks and maximizing the sum of the log-likelihood of a set of vertices for each walk.



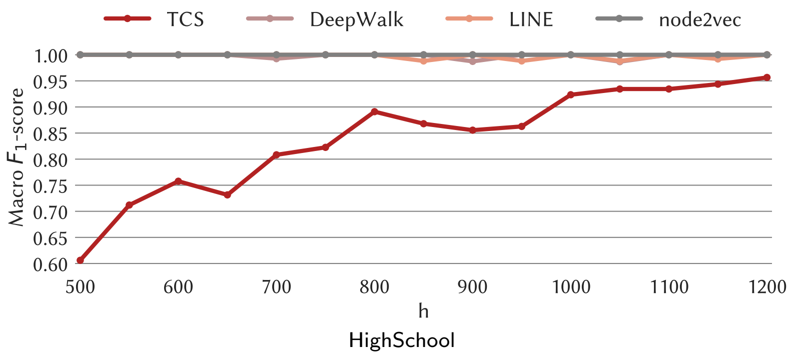


Figure 7.5: Graph classification: Macro F_1-score of the proposed temporal-community-search-based graph-embedding method TCS and the competing methods, with varying the dimensionality h of the output embeddings, on the PrimarySchool and HighSchool datasets.

- LINE [224], which optimizes a suitable objective function preserving both first-order (one-hop) and second-order (two-hop) proximities. Neighborhoods are not explored via random walk, but in a breadth-first fashion.

- node2vec [118], which is based on the same idea underlying DeepWalk, but allowing more flexibility on how random walks explore and leave the neighborhood of the current vertex.

These three methods consider non-temporal graphs. Therefore, we feed them with aggregated graphs in which every edge exists if it exists in at least one timestamp. We tune the parameters p and q of node2vec as in the original paper [118], i.e., by performing a grid search with $p, q \in \{0.25, 0.50, 1, 2, 4\}$, while keeping the dimensionality of the embeddings fixed to $h = 200$ and $h = 625$, for the PrimarySchool and HighSchool datasets, respectively. The other competing methods, DeepWalk and LINE, and our method based on temporal community search (which we refer to as TCS in the following) do not have parameters (apart from the dimensionality h of the output embeddings). After filtering out those vertices representing the teachers, we partition the remaining vertices (i.e., the students) into training and test sets with an 80-20 split. A standard scaler is applied to the features extracted by each embedding method and, then, a penalized logistic-regression classifier is trained.

In Figure 7.5 we report classification results in terms of Macro F_1-score, with varying the dimensionality h of the embeddings. On the PrimarySchool dataset, for $h \geq 200$, our TCS has performance close to 1 in terms Macro F_1-score, similarly to the three baselines. It can be observed that the TCS results are better as h gets higher; in particular, TCS is even better than node2vec for $h = |T|$. This is expected and is motivated as, for higher h, TCS is allowed to rely on more temporal information about the vertices. On the HighSchool dataset, TCS is outperformed by all methods for smaller h. However, again, the performance of TCS becomes competitive for larger h, up to achieving comparable results to the best method(s) for $h = |T|$.

7.4 Summary

In this chapter we show the usefulness of notions introduced in Chapter 4, i.e., span-cores, maximal span-cores, and temporal community search, in multiple analyses and applications to face-to-face interaction networks gathered in schools. The contributions of this work are the following:

- we derive interesting temporal patterns of groups of students from (maximal) span-cores, i.e., daily activity patterns, mixing patterns, and interaction length;
- we devise a simple yet effective procedure to detect anomalous edges and intervals in a temporal network;
- we show how to build a simple graph-embedding technique that makes use of the temporal information provided by temporal community search and we apply it to the task of vertex classification.

Chapter 8

Visualizing structural balance in signed networks

As described in Chapter 5, signed networks are network representations in which edges are annotated as positive or negative [121]. They have been applied in a large variety of domains in which interactions between entities are either friendly or antagonistic, e.g., anthropology [120], political debates [144, 66], international relations [62, 71], and online social media and social networks [223]. The theory of *structural balance* has established as the standard for studying, from a theoretical standpoint in sociology and psychology, the formation of opinions in both individuals and social groups [4, 124]. Structural balance is widely applied to signed networks, e.g., for the analysis of social media [158], the understanding of opinion dynamics [192], and the study of opinion separation [240]. A signed network has been proved to be *structurally balanced* or *balanced* if and only if all cycles are balanced, i.e., include an even number of negative edges [52]. As a consequence, network's nodes can be assigned to two different sets such that we find only positive ties between nodes in the same set and all negative ones between nodes of different sets [73], resembling the definition of polarized communities introduced in Chapter 5. Figure 8.1 shows two simple examples of balanced and unbalanced networks. The network on the left is balanced and has the two properties discussed above, i.e., all cycles are balanced and a clustering can be found in agreement to all edges' signs. On the other hand, the network on the right is not balanced: there are unbalanced cycles (e.g., the one composed by the node sequence $[A, B, D, C, A]$) and there are edges disagreeing with the clustering (e.g., edge (B, E)). Even if a balanced network represents the most natural configuration, structural balance is not necessarily a "positive" configuration, e.g., it is observed in the alliance network between European nations just before World War I [194]. Moreover, most of the large real-world networks are expected to be unbalanced since a single unbalanced cycle makes the whole network unbalanced. Therefore, it has also been shown the importance of measuring to what extent an unbalanced signed network is close to be balanced [152]. Structural balance is also linked with *group polarization*, i.e., the division of a group of entities (e.g., nodes of a network) into two subgroups each reaching consensus and having opposite opinions [235, 74, 45].

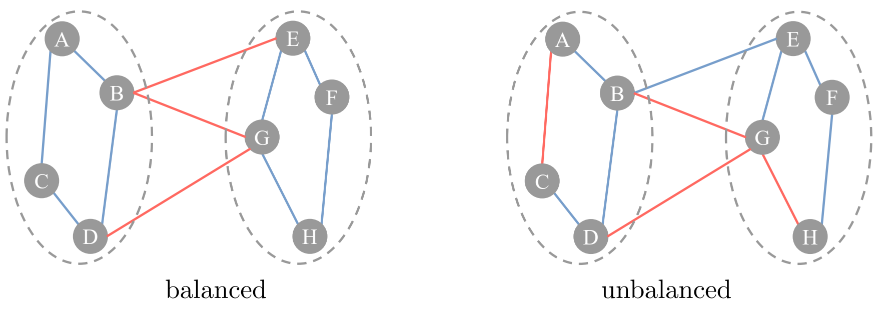


Figure 8.1: Examples of balanced (left) and unbalanced (right) networks. Positive edges are reported in blue, while negative edges in red.

Network visualization has emerged as a key complement to standard network analysis techniques to fill the gap between computation and interpretation, communicate findings, and deepen insight [150]. A large variety of network layouts exists in literature [138, 93, 69, 125, 140] and, also, implemented for visualization application, as, e.g., Gephi [28] and Cytoscape [212]. Surprisingly, little attention has been paid to the visualization of signed networks [48, 152] and, to our knowledge, none of the existing layouts highlights structural balance properties of signed networks.

In this chapter we tackle the task of identifying, through a visualization, whether a connected signed network is balanced or unbalanced and, in the latter case, how much the network is unbalanced. The proposed visualization method, Structural-balance-viz, places nodes in a Cartesian coordinate system exploiting spectral properties of the signed Laplacian matrix, borrowing theoretical intuitions similar to Chapter 5 Edges are colored and bundled to make positive and negative signs distinguishable and to ease the understanding of the global balance/polarization of the network. At a glance, it is possible to catch if a network is balanced: no positive edges cross the y-axis and no negative edges have both endpoints in the same quadrant, namely, the y-axis finds a partition of the nodes as explained in [73]. The visual perception of the portion of edges "disagreeing" with the partitioning, i.e., the fraction of positive edges crossing the y-axis and negative edges internal to a quadrant, gives an indication of the level of balance of a network. Moreover, we utilize the x-axis as a scale to show cumulative characteristics of the sets of nodes identified by the y-axis, and include a textual indication of the level of balance of the network under analysis in order to improve the comparability between different visualizations.

The layout produced by Structural-balance-viz has the following characteristics that are useful in a variety of network analysis tasks: (i) it shows whether the input network is balanced or not and, in the second case, how close the network is to be balanced; (ii) by nodes' x-coordinate, it provides an indication of the contribute to the balance structure of the network and, also, of the individual balance/polarization of each node (such information might be exploited, e.g., for the task of finding non-polarized representatives [186]); (iii) it identifies two factions of nodes on the basis of their polarization which finds applications in clustering problems, e.g., 2-correlation-clustering [60, 14]; (iv) the scale represented by the x-axis shows cumulative characteristics of the identified factions, e.g., size or internal clustering coefficient; and, (v) the resulting visualization are reproducible (desirable feature but not common to all network layouts, e.g., force based) and easy to compare in terms of balance structure. We verify such characteristics by running Structural-balance-viz on synthetic networks and a real-world dataset representing political debates.

8.1 Structural-balance-viz

First, we provide preliminary notations and definitions. We denote a signed undirected network as $G = (V, E_+, E_-)$, where V is a set of nodes, E_+ is a set of positive edges, and E_- is a set of negative edges. In this work, we require G to be connected. Let A be the signed adjacency matrix of G, i.e., for each pair of nodes $u, v \in V$, $A[u, v] = 1$ if $(u, v) \in E_+$, $A[u, v] = -1$ if $(u, v) \in E_-$, $A[u, v] = 0$ otherwise. Let also $\bar{D} = \text{diag}(\bar{d}_{u_1}, \dots, \bar{d}_{u_{|V|}})$ be the signed degree matrix of G, where $\bar{d}_u = \sum_{v \in V} |A[u, v]|$ represents the signed degree, i.e., the number of neighbors disregarding the sign, of a node $u \in V$. Finally, we define the signed Laplacian matrix of G as:

$$\bar{L} = \bar{D} - A. \tag{8.1}$$

We now describe our algorithm for visualizing structural balance in signed networks, which is outlined as Algorithm 8.1. As mentioned beforehand, Structural-balance-viz makes use of the signed Laplacian of the input network G. In fact, it starts by computing the signed Laplacian together with its smallest eigenvalue λ_m and the corresponding eigenvector $\vec{v}_m$ (Line 2). At this point, we already have all the information required for the visualization handy. At first, we identify the coordinates of the nodes in V and store them in $\mathbf{X}$ and $\mathbf{Y}$ (cycle starting at Line 4). The x-coordinate of each node u is directly obtained by the element of $\vec{v}_m$ corresponding to u. Since more than a node might have the same abscissa and we want to avoid nodes to overlap, the y-coordinates are computed in order to distribute nodes having the same x-coordinate vertically. Next (Lines 7 - 10), edges are divided into four sets since, on the basis of the coordinates of their endpoints and of their sign, different layouts are applied:

Algorithm 8.1: Structural-balance-viz

```
Input: A signed network $G = (V, E_+, E_-)$ and a network measure $\mu$ (optional).
Output: A visualization of $G$.
/* Eigenvalue decomposition                                              */
1  compute the signed Laplacian $\bar{L}$ of $G$
2  compute the smallest eigenvalue $\lambda_m$ of $\bar{L}$ and its corresponding eigenvector $\vec{v}_m$
/* Nodes coordinates                                                     */
3  $\mathbf{X} \leftarrow \emptyset$;  $\mathbf{Y} \leftarrow \emptyset$
4  forall $u \in V$ do
5      $\mathbf{X}[u] = \vec{v}_m[u]$
6      $\mathbf{Y}[u] = |\{v \in V \mid \vec{v}_m[v] = \vec{v}_m[u] \wedge v < u\}|$
/* Edge partitioning                                                     */
7  $E^i_+ = \{e = (u, v) \in E_+ \mid \mathbf{X}[u] = \mathbf{X}[v]\}$
8  $E^i_- = \{e = (u, v) \in E_- \mid \mathbf{X}[u] = \mathbf{X}[v]\}$
9  $E^e_+ = E_+ \setminus E^i_+$
10 $E^e_- = E_- \setminus E^i_-$
/* Drawing                                                               */
11 draw the Cartesian axes
12 draw the nodes in $V$ according to $\mathbf{X}$ and $\mathbf{Y}$
13 draw the edges in $E^i_+$ in blue with horizontal-external bundling
14 draw the edges in $E^i_-$ in red with horizontal-internal bundling
15 draw the edges in $E^e_+$ in blue with vertical-upper bundling
16 draw the edges in $E^e_-$ in red with vertical-lower bundling
/* Additional features                                                   */
17 if $\mu \neq$ NULL then
18     $C_l = \{u \in V \mid \mathbf{X}[u] < 0\}$;  $C_r = \{u \in V \mid \mathbf{X}[u] \geq 0\}$
19     let $\gamma = \mu(C_l) - \mu(C_r)$ be the angular coefficient of the $x$-axis
20 draw the label "$y = \lambda_m$"
```

- E^i_+ contains the positive edges having two endpoint with the same x-coordinate;
- E^i_- contains the negative edges having two endpoint with the same x-coordinate;
- E^e_+ contains the positive edges having two endpoint with different x-coordinate;
- E^e_- contains the negative edges having two endpoint with different x-coordinate.

Structural-balance-viz is then ready to draw the visualization (Lines 11 - 16). At first, the Cartesian axes and the nodes are positioned. Then, the edges are drawn exploiting coloring and bundling to highlight their sign. In particular, positive edges are depicted in blue, while negative edges in red. A positive edge $e_+ \in E_+$ is bundled towards the top of the visualization, if $e_+ \in E^e_+$, or externally, if $e_+ \in E^i_+$; while a negative edge $e_- \in E_-$ is bundled towards the bottom, if $e_- \in E^e_-$, or internally, if $e_- \in E^i_-$.

In order to improve the informativeness of our layout, we include two additional features in Structural-balance-viz (from Line 17): one wants to provide information about the two sets of nodes identified by the y-axis, while the latter has the aim of making different visualizations more comparable.

Any eigenvector $\vec{v}$ of the signed Laplacian can be used to derive a partition of network's nodes into two sets on the basis of the sign of the corresponding elements in $\vec{v}$. Such partitioning is at the basis of spectral-clustering methods [61] and it can identify polarized structures, i.e., two sets of nodes showing high internal consensus and warring between each other [45, 100, 59]. In the proposed visualization, the two sets are identified by the nodes in the left and in the right quadrants, i.e., C_l and C_r computed at Line 18 of Structural-balance-viz, respectively. In practical applications, it is often of interest to know (and visualize) network measures of the two polarized sets, e.g., size, internal clustering coefficient, internal density of positive edges, ratio of positive

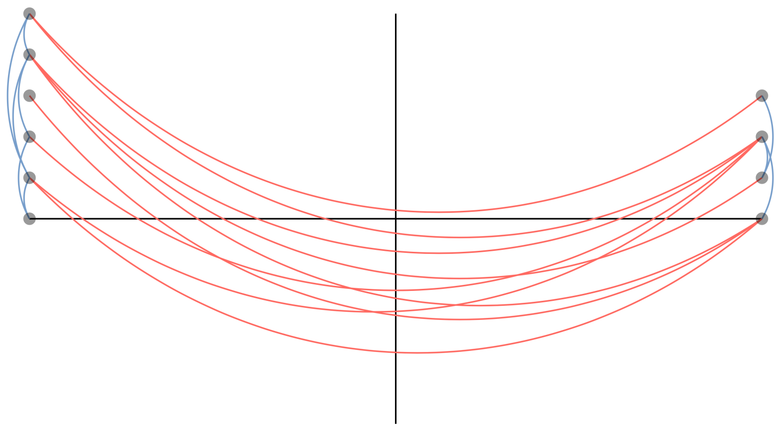


Figure 8.2: Visualization by Structural-balance-viz of a balanced network: all the cycles are balanced.

edges, etc. We provide a simple visual expedient based on the angular coefficient of the x-axis that resembles the behavior of a scale. Let μ be the network measure of interest. Note that μ is an optional input parameter of Structural-balance-viz and the lines corresponding to this additional feature are executed if μ is actually provided in input. We define the angular coefficient of the x-axis as

$$\gamma = \mu(C_l) - \mu(C_r). \tag{8.2}$$

The work enclosed in [128, 129] proves theoretical bounds on the smallest eigenvalue of the Laplacian of a signed network and investigates its relationship with respect to the level of balance in the network. It is shown that a connected signed network is structurally balanced if and only if $\lambda_m = 0$, i.e., the smallest eigenvalue of the Laplacian is zero, and that the higher λ_m, the lower the level of balance of the network is. Therefore, λ_m is the simplest indicator to take into account for comparing structural balance in different networks (of equal densities). More complex indicators of balance could also be employed [15]. Ideally, the y-coordinate where the x-axis crosses the y-axis would be a simple manner to graphically show λ_m. Unfortunately, we devoted consistent effort to visualize such information in this way, but all attempts worsened the clarity of the layout (e.g., cut off edges). To this extent, we include in Structural-balance-viz a label reporting the value of λ_m on the top of the y-axis and leave the visualization of λ_m without the label as future work.

The time complexity of Structural-balance-viz is governed by the time required by the eigenvalue decomposition of $\bar{L}$, while the space complexity is $\mathcal{O}(|V|^2)$, again imposed by $\bar{L}$. Note that computational-intensive network measures μ might considerably extend the running time when drawing large networks.

Figures 8.2 and 8.3 show two examples of visualizations generated by Structural-balance-viz for a balanced and an unbalanced network, respectively. For such visualizations, we remove the label reporting λ_m to prove how obvious the difference between the two networks is even without textual information. Also, as for all other examples in this chapter, edge bundling is not applied. It is immediate to note that the network represented in Figure 8.2 is balanced: all the nodes are at the extremes of the x-axis and no blue (red) edge crosses the y-axis (lays in the same quadrant). This configuration highlights the fact that all the cycles of the represented network are balanced. On the other hand, Figure 8.3 shows an unbalanced network since there are positive edges in-between the two factions of nodes and a negative edge within two nodes in the left quadrant; therefore, we easily find the presence of unbalanced cycles.

Figure 8.4 shows the same network of Figure 8.3 with both the additional features of Structural-balance-viz; in this case, the x-axis scale compares the size of the two factions of nodes, i.e., μ counts the number of nodes in the sets. At a glance, it is possible to understand that the left faction is slightly larger than the right one (six and four nodes, respectively) and that the smallest eigenvalue of the signed Laplacian is not far from zero; this means that the network is not far from being balanced (i.e., there are not many unbalanced cycles).

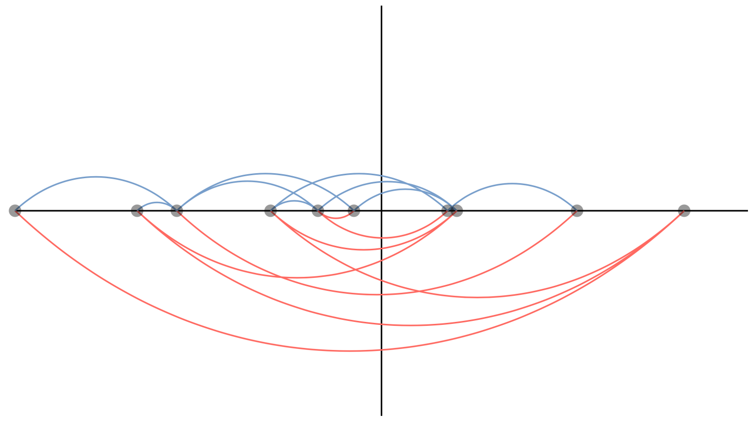


Figure 8.3: Visualization by Structural-balance-viz of an unbalanced network: not all the cycles are balanced.

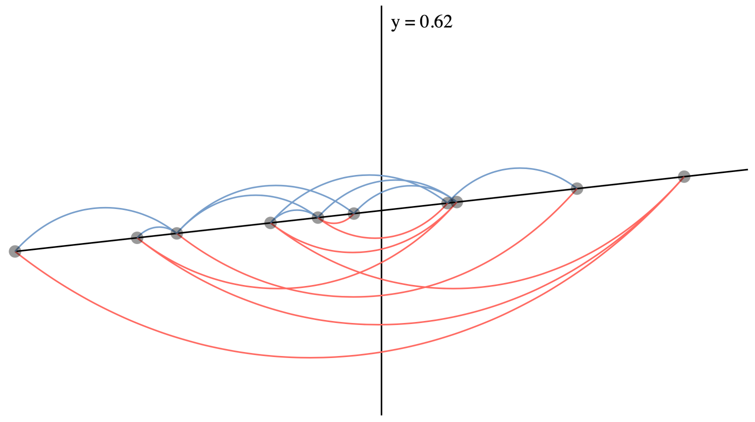


Figure 8.4: Visualization by Structural-balance-viz of an unbalanced network (same as Figure 8.3) with the the additional features. The x-axis scale compares the size of the two factions of nodes.

8.2 Validation and application

In this section we validate the proposed network layout by visualizing synthetic networks. Also, we apply Structural-balance-viz to derive concrete insights from the Congress dataset, the same introduced in Chapter 5, representing political debates.

We develop Structural-balance-viz by using D3.js with a Java back-end. The visualization is made available by a web interface that allows the selection of the input dataset and of μ (i.e., the network measure that defines the angular coefficient of the x-axis)[1]. The current implementation can consider only the size of the sets of nodes as μ, but the code is easily extendable to consider other characteristics. The time required by our implementation to produce each visualization has always been less than a few seconds.

8.2.1 Validation: synthetic networks

We first focus our attention on synthetic-generated networks with the aim of proving that the visualizations produced by Structural-balance-viz are easily comparable. The generative process for signed networks we follow requires in input three parameters: n indicates the number of nodes, δ defines the edge density, while ν is the ratio of unbalanced triangles in the network (which is another indicator of how much a network is balanced [73]). The procedure works as follows:

[1]Code available at github.com/egalimberti/balance_visualization.

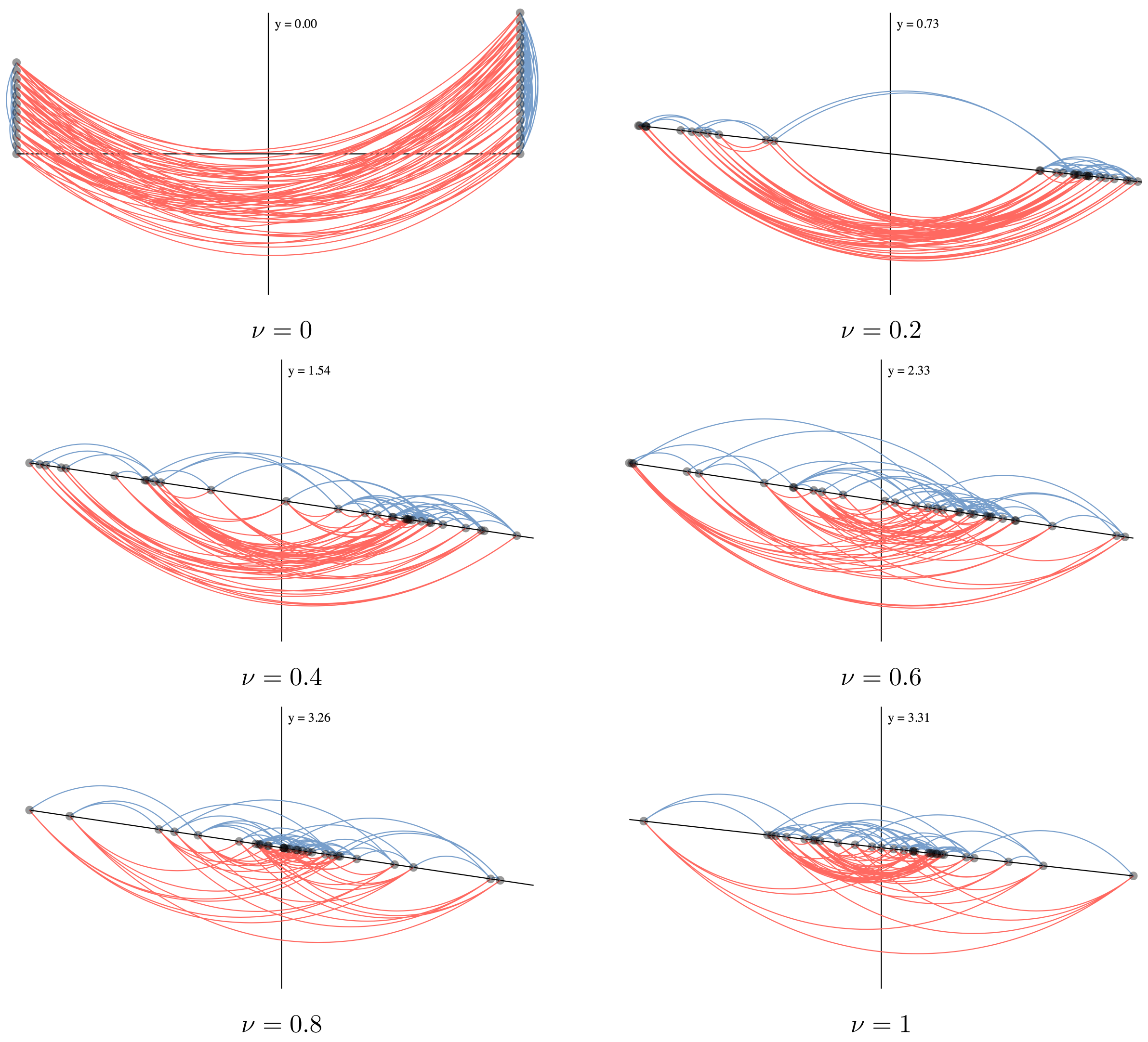


Figure 8.5: Visualization by Structural-balance-viz of synthetic networks for increasing values of ν ($n = 30$, $\delta = 0.3$).

- generate a complete balanced network of n nodes (this can be achieved by partitioning the n nodes into two and then assigning negative sign to the edges connecting nodes in different sets while positive sign to all others edges);
- randomly remove edges that do not disconnect the network until the edge density is less or equal than δ;
- randomly change signs of edges appearing in balanced triangles until the ratio of unbalanced triangles is less or equal than ν.

In Figure 8.5 we report our visualization for six networks generated by the described procedure by progressively increasing ν ($\nu \in [0, 0.2, 0.4, 0.6, 0.8, 1]$) while keeping n and δ fixed ($n = 30$, $\delta = 0.3$). Therefore, we have the full range of networks in terms of structural balance: on one extreme ($\nu = 0$) the network is perfectly balanced, on the other ($\nu = 1$) the network has no balanced triangles. When $\nu = 0$, as expected, we obtain the perfectly distinguishable configuration of balanced networks, where all nodes are in either extremes of the x-axis, no positive edge crosses the y-axis, and no negative edge entirely lies in the same quadrant. Note that, for the balanced case, we do not provide in input to Structural-balance-viz any network function μ since the number of nodes in the sets can be inferred by the height of the two stacks. As ν grows, the most of

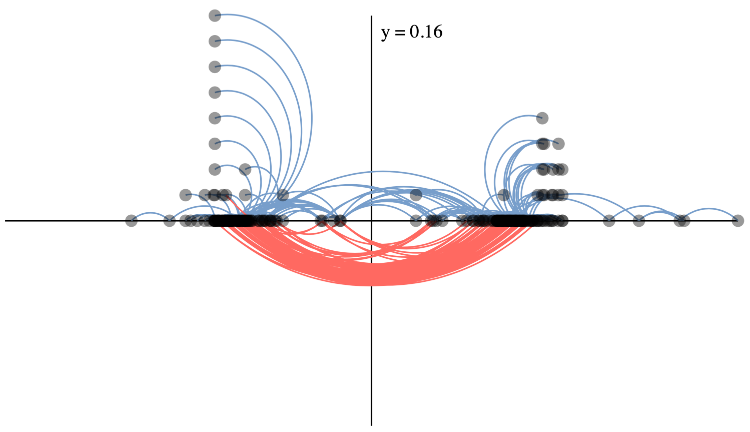


Figure 8.6: Visualization by Structural-balance-viz of the United States Congress network.

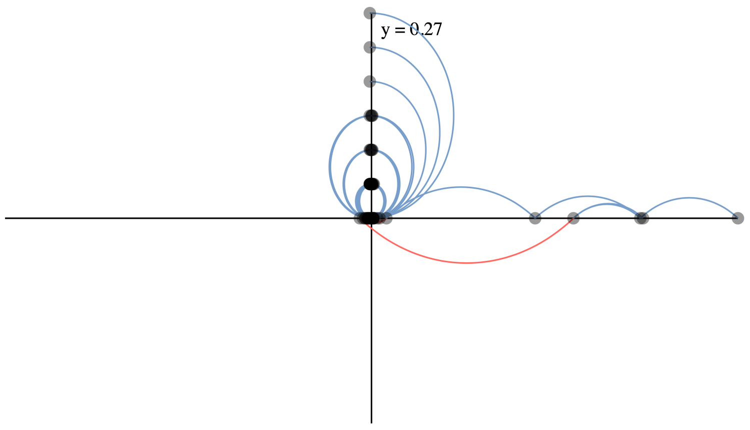


Figure 8.7: Visualization by Structural-balance-viz of the United States Congress network after sign reshuffling.

the nodes gradually moves from the extreme ordinates to the center of the plot; nonetheless, even for $\nu = 1$, we note a few highly-polarized nodes at the margins of the horizontal domain. In addition, more and more both positive edges cross the y-axis and negative edges are within one of the two quadrants. The additional features result to be extremely useful in these cases. At first, the scale gives a precise indication that the right faction is larger than the left one for all values of ν. Also, the smallest eigenvalue of the signed Laplancian, which grows coherently with ν, eases the comparison of visualizations that might appear similar (e.g., $\nu = 0.8$ and $\nu = 1$) and provides a definitive indication about the structural balance of the visualized networks.

8.2.2 A case study: the United States Congress network

Next, we apply Structural-balance-viz to the analysis of the Congress dataset[2], a real-world network modeling political debate in the United States Congress. Nodes ($|V| = 219$) are politicians speaking in the Congress, edges ($|E_+ \cup E_-| = 521$) denote that a speaker mentions another speaker, while signs report whether mentions are in support (positive) or opposition (negative).

Figure 8.6 shows the visualization of the original Congress network. It is easy to notice that the members are divided into two (almost) equally-sized factions that are close to be balanced; in fact, there is only one negative edge within the left faction and a relatively few positive edges crossing the y-axis. The x-axis can be seen as the left-right political spectrum: the most of the

[2]Dataset available at konect.cc

politicians are quite moderate, while there are some polarized members especially in the right, and a few nodes that lay close $x = 0$ (probably the mediators between the two factions).

To have a better understanding of the structural balance of the the Congress network, we compare it to a null model. In particular, we maintain the same network structure while reshuffling the edge signs, leaving the number of positive and negative edges unchanged. The visualization of the resulting reshuffled network is reported in Figure 8.7. In this case, the balance/polarization structure of the network is destroyed since the majority of the nodes collapse close to the origin. All the negative edges (except one) lay between such nodes and are no more visible in the layout. Only five members maintain their polarization in the right. Moreover, the smallest eigenvalue of the signed Laplacian is greater than in the original network. All this indications suggests that, the United States Congress network is more balanced/polarized than what is expected by chance, according to a reshuffled null model. The Congress is instead quite polarized, very close to being structurally balanced, due to the political parties and alliances.

8.3 Summary

In this chapter we introduce Structural-balance-viz: a novel algorithm that places nodes in a Cartesian coordinate system, that resembles the behavior of a scale, and exploits edge coloring and bundling for showing whether a connected signed network is balance or unbalanced and, in the latter case, how far it is from being balanced. Structural-balance-viz is validated by the analysis of synthetic networks: it is proved to provide an indication of balance/polarization of the whole network and individually of each node, to identify two factions of nodes on the basis of their polarization and show their cumulative characteristics, and to produce reproducible and easily comparable visualizations. A direct application to a real-world dataset about political debates confirms that Structural-balance-viz is able to provide meaningful insights about the balance/polarization structure of the network.

Chapter 9

Conclusions

This thesis expands the bulk of literature about complex networks with two main contributions. The first part proposes a series of novel definitions and algorithms for finding dense structures in multilayer, temporal, and signed networks that enrich the standard network representation with additional features. In particular, we define core decomposition in multilayer networks and we show how it can be employed in a bunch of theoretical applications; we introduce core decomposition in temporal networks which provides a set of dense structures associated with a clear temporal collocation; and, we show how the problem of finding polarized communities in signed networks can be tackled by algorithms based on spectral theory. In the second part of the thesis, instead, we focus on the in-depth study of the substructure in temporal and signed networks derived from real-world data. First, we study the migration of researchers around the globe from a complex network perspective; then, we confirm that span-cores are a valuable tool for the analysis of face-to-face interaction networks; and, finally, we show that proper network-drawing methods are powerful for unveiling insight about the balance/polarized structure of signed networks.

Future work

Each contribution of the current thesis opens enticing avenues for further inquiry:

- Multilayer core decomposition might be employed for the analysis of multilayer brain networks in which each layer represents a patient, vertices are brain regions, and edges are co-activation interactions measured by fMRI scans. In this scenario, multilayer core decomposition might result to be a powerful tool to identify common substructures to patients affected by diseases or under the assumption of drugs and, also, to select features in order to discriminate actual patients from healthy individuals.

- It would be of interest to study the role of maximal span-cores in spreading processes on temporal networks. Also, span-cores represent features that can be used for network fingerprinting and classification as well as for model validation, and that could provide support for new ways of visualizing large-scale time-varying graphs.

- The application of the proposed definition of 2-Polarized-Communities to real-world networks with positive and negative relationships can have implications in computational social science problems. For instance, understanding opinion shifts in data streaming from social media sources can be investigated in terms of polarized communities. Opinions shared within vertices (individuals) belonging to the same community are likely to be reinforced after different interactions; discussions within individuals with antagonistic perspectives may result in both opinion shifts and controversy amplification. Thus, it would be interesting to study extensions of the 2-Polarized-Communities problem in the setting of temporal networks.

- The analysis carried out about the scientific migration network can be expanded by studying the correlation between hub and authority scores with respect to metrics of research/academic success and economic indicators, e.g., GDP; also, the analysis might be restricted to a specific geographical region (e.g., Europe) to study migrations at smaller granularity (e.g., cities) or according to specific science fields; moreover, it would be of interest to replicate our analysis on other datasources to confirm/integrate the obtained results results.

- The implementation of Structural-balance-viz can be deployed to a public web interface and made available for network visualization tools, e.g., Cytoscape, so that practitioners and researchers could use it for visualizing the polarized structure of real-world signed networks.

Bibliography

[1] J. Abello, M. G. Resende, and S. Sudarsky. Massive quasi-clique detection. In *LATIN 2002: Theoretical Informatics*, pages 598–612. Springer, 2002.

[2] A. Agrawal, D. Kapur, J. McHale, and A. Oettl. Brain drain or brain bank? the impact of skilled emigration on poor-country innovation. *Journal of Urban Economics*, 69(1):43–55, 2011.

[3] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):23, 2008.

[4] I. Ajzen. Nature and operation of attitudes. *Annual review of psychology*, 52(1):27–58, 2001.

[5] J. Akiyama, D. Avis, V. Chvátal, and H. Era. Balancing signed graphs. *Discrete Applied Mathematics*, 3(4):227–233, 1981.

[6] L. Akoglu, D. H. Chau, C. Faloutsos, N. Tatti, H. Tong, J. Vreeken, and L. Tong. Mining connection pathways for marked nodes in large graphs. In *SDM*, 2013.

[7] J. I. Alvarez-hamelin, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *NIPS*, 2006.

[8] L. A. Amaral and J. M. Ottino. Complex systems and networks: challenges and opportunities for chemical and biological engineers. *Chemical Engineering Science*, 59(8-9):1653–1666, 2004.

[9] P. Anchuri and M. Magdon-Ismail. Communities and balance in signed networks: A spectral approach. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 235–242, 2012.

[10] R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 25–37. Springer, 2009.

[11] R. Andersen and K. J. Lang. Communities from seed sets. In *WWW*, 2006.

[12] A. Angel, N. Sarkas, N. Koudas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *PVLDB*, 5(6), 2012.

[13] T. Antal, P. Krapivsky, and S. Redner. Social balance on networks: The dynamics of friendship and enmity. *Physica D*, 224(130), 2006.

[14] S. Aref, A. J. Mason, and M. C. Wilson. Computing the line index of balance using integer programming optimisation. In *Optimization Problems in Graph Theory*, pages 65–84. Springer, 2018.

[15] S. Aref and M. C. Wilson. Measuring partial balance in signed networks. *Journal of Complex Networks*, 6(4):566–595, 2017.

[16] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *STOC*, 1995.

[17] Y. Asahiro, R. Hassin, and K. Iwama. Complexity of finding dense subgraphs. *Discr. Ap. Math.*, 121(1-3), 2002.

[18] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. *Journal of Algorithms*, 34(2):203–221, 2000.

[19] N. Azimi-Tafreshi, J. Gómez-Gardenes, and S. Dorogovtsev. k- core percolation on multiplex networks. *Physical Review E*, 90(3):032816, 2014.

[20] G. D. Bader and C. W. V. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4:2, 2003.

[21] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5):454–465, 2012.

[22] O. D. Balalau, F. Bonchi, T. Chan, F. Gullo, and M. Sozio. Finding subgraphs with maximum total density and limited overlap. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 379–388. ACM, 2015.

[23] D. Baldassarri and P. Bearman. Dynamics of political polarization. *American sociological review*, 72(5):784–811, 2007.

[24] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine learning*, 56(1-3):89–113, 2004.

[25] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo. Efficient and effective community search. *DAMI*, 29(5), 2015.

[26] A. Barrat, M. Barthélemy, and A. Vespignani. *Dynamical processes on complex networks*. Cambridge University Press, Cambridge, 2008.

[27] O. Bastert and C. Matuszewski. Layered drawings of digraphs. In *Drawing graphs*, pages 87–120. Springer, 2001.

[28] M. Bastian, S. Heymann, and M. Jacomy. Gephi: an open source software for exploring and manipulating networks. In *Third international AAAI conference on weblogs and social media*, 2009.

[29] V. Batagelj, A. Mrvar, and M. Zaversnik. Partitioning approach to visualization of large graphs. In *Int. Symp. on Graph Drawing*, pages 90–97, 1999.

[30] V. Batagelj and M. Zaveršnik. Fast algorithms for determining (generalized) core groups in social networks. *Advances in Data Analysis and Classification*, 5(2):129–145, 2011.

[31] A. Bavelas. Communication patterns in task-oriented groups. *The Journal of the Acoustical Society of America*, 22(6):725–730, 1950.

[32] G. Beigi, J. Tang, and H. Liu. Signed link analysis in social media networks. In *International AAAI Conference on Web and Social Media (ICWSM)*, pages 539–542, 2016.

[33] R. Bellman. On the approximation of curves by line segments using dynamic programming. *Commun. ACM*, 4(6):284–, 1961.

[34] A. Belyi, I. Bojic, S. Sobolevsky, I. Sitko, B. Hawelka, L. Rudikova, A. Kurbatski, and C. Ratti. Global multi-layer network of human mobility. *International Journal of Geographical Information Science*, 31(7):1381–1402, 2017.

[35] M. Bentert, A.-S. Himmel, H. Molter, M. Marik, R. Niedermeier, and R. Saitenmacher. Listing all maximal k-plexes in temporal graphs. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 41–46. IEEE, 2018.

[36] M. Berlingerio, F. Bonchi, B. Bringmann, and A. Gionis. Mining graph evolution rules. In *ECML PKDD 2009*.

[37] M. Berlingerio, M. Coscia, and F. Giannotti. Finding redundant and complementary communities in multidimensional networks. In *CIKM*, pages 2181–2184, 2011.

[38] S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. Tsourakakis. Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 173–182. ACM, 2015.

[39] Y. Bian, Y. Yan, W. Cheng, W. Wang, D. Luo, and X. Zhang. On multi-query local community detection. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 9–18. IEEE, 2018.

[40] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[41] B. Boden, S. Günnemann, H. Hoffmann, and T. Seidl. Mining coherent subgraphs in multi-layer graphs with edge labels. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1258–1266. ACM, 2012.

[42] P. Bogdanov, M. Mongiovì, and A. K. Singh. Mining heavy subgraphs in time-evolving networks. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 81–90. IEEE, 2011.

[43] J. Bohannon and K. Doran. Introducing orcid, 2017.

[44] F. Bonchi, I. Bordino, F. Gullo, and G. Stilo. Identifying buzzing stories via anomalous temporal subgraph discovery. In *WI 2016*, 2016.

[45] F. Bonchi, E. Galimberti, A. Gionis, B. Ordozgoiti, and G. Ruffo. Discovering polarized communities in signed networks. In *Proceedings of the 2019 ACM on Conference on Information and Knowledge Management*. ACM, 2019.

[46] F. Bonchi, A. Gionis, F. Gullo, C. E. Tsourakakis, and A. Ukkonen. Chromatic correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 9(4):34, 2015.

[47] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *KDD*, pages 1316–1325, 2014.

[48] U. Brandes, D. Fleischer, and J. Lerner. Summarizing dynamic bipolar conflict structures. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1486–1499, 2006.

[49] B. Bringmann, M. Berlingerio, F. Bonchi, and A. Gionis. Learning and predicting the evolution of social networks. *IEEE Intelligent Systems*, 25(4):26–35, 2010.

[50] J. Brundidge. Encountering 'difference' in the contemporary public sphere: The contribution of the internet to the heterogeneity of political discussion networks. *Journal of Communication*, 60(4):680–700, 2010.

[51] D. Cai, Z. Shao, X. He, X. Yan, and J. Han. Community mining from multi-relational networks. In *PKDD*. 2005.

[52] D. Cartwright and F. Harary. Structural balance: a generalization of heider's theory. *Psychological review*, 63(5):277, 1956.

[53] R. Cerqueti, G. P. Clemente, and R. Grassi. A network-based measure of the socio-economic roots of the migration flows. *Social Indicators Research*, pages 1–18, 2018.

[54] N. Cesa-Bianchi, C. Gentile, F. Vitale, and G. Zappella. A correlation clustering approach to link classification in signed networks. In *Annual Conference on Learning Theory*, volume 23, pages 34–1, 2012.

[55] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 84–95. Springer, 2000.

[56] M. Charikar, Y. Naamad, and J. Wu. On finding dense common subgraphs. *arXiv preprint arXiv:1802.06361*, 2018.

[57] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu. Efficient core decomposition in massive networks. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 51–62. IEEE, 2011.

[58] K.-Y. Chiang, J. J. Whang, and I. S. Dhillon. Scalable clustering of signed networks using balance normalized cut. In *Proceedings of the 21st ACM international conference on Information and knowledge management (CIKM)*, pages 615–624, 2012.

[59] L. Chu, Z. Wang, J. Pei, J. Wang, Z. Zhao, and E. Chen. Finding gangs in war from signed networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1505–1514. ACM, 2016.

[60] T. Coleman, J. Saunderson, and A. Wirth. A local-search 2-approximation for 2-correlation-clustering. In *European Symposium on Algorithms*, pages 308–319, 2008.

[61] T. Coleman, J. Saunderson, and A. Wirth. Spectral clustering with inconsistent advice. In *Proceedings of the 25th international conference on Machine learning*, pages 152–159. ACM, 2008.

[62] M. J. Crescenzi. Reputation and interstate conflict. *American Journal of Political Science*, 51(2):382–396, 2007.

[63] F. Cribari-Neto, N. L. Garcia, and K. L. Vasconcellos. A note on inverse moments of binomial variates. *Brazilian Review of Econometrics*, 20(2):269–277, 2000.

[64] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.

[65] A. Das Sarma, A. Jain, and C. Yu. Dynamic relationship and event discovery. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 207–216. ACM, 2011.

[66] W. De Nooy and J. Kleinnijenhuis. Polarization in the media during an election campaign: A dynamic network model predicting support and attack among political actors. *Political Communication*, 30(1):117–138, 2013.

[67] E. Desmier, M. Plantevit, C. Robardet, and J.-F. Boulicaut. Cohesive co-evolution patterns in dynamic attributed graphs. In *International Conference on Discovery Science*, pages 110–124. Springer, 2012.

[68] P. Deville, D. Wang, R. Sinatra, C. Song, V. D. Blondel, and A.-L. Barabási. Career on the move: Geography, stratification, and scientific impact. *Scientific reports*, 4:4770, 2014.

[69] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, 4(5):235–282, 1994.

[70] M. E. Dickison, M. Magnani, and L. Rossi. *Multilayer Social Networks.* Cambridge University Press, 2016.

[71] P. Doreian and A. Mrvar. Structural balance and signed international relations. *Journal of Social Structure*, 16:1, 2015.

[72] X. Du, R. Jin, L. Ding, V. E. Lee, and J. H. Thornton, Jr. Migration motif: a spatial - temporal pattern mining approach for financial markets. In *KDD*, 2009.

[73] D. Easley, J. Kleinberg, et al. *Networks, crowds, and markets*, volume 8. Cambridge University Press, Cambridge, Massachusetts, 2010.

[74] D. Easley, J. Kleinberg, et al. Networks, crowds, and markets: Reasoning about a highly connected world. *Significance*, 9:43–44, 2012.

[75] M. Eidsaa and E. Almaas. s-core network decomposition: A generalization of k-core analysis to weighted networks. *Phys. Rev. E*, 88:062819, Dec 2013.

[76] A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In *Proceedings of the 24th International Conference on World Wide Web*, pages 300–310. International World Wide Web Conferences Steering Committee, 2015.

[77] A. Epasto and B. Perozzi. Is a single embedding enough? learning node representations that capture multiple social contexts. In *The World Wide Web Conference*, pages 394–404. ACM, 2019.

[78] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation*, pages 403–414. Springer, 2010.

[79] P. Érdi, K. Makovi, Z. Somogyvári, K. Strandburg, J. Tobochnik, P. Volf, and L. Zalányi. Prediction of emerging technologies based on analysis of the us patent citation network. *Scientometrics*, 95(1):225–242, 2013.

[80] J.-M. Esteban and D. Ray. On the measurement of polarization. *Econometrica: Journal of the Econometric Society*, pages 819–851, 1994.

[81] G. Fagiolo and M. Mastrorillo. International migration network: Topology and modeling. *Physical Review E*, 88(1):012812, 2013.

[82] G. Fagiolo and G. Santoni. Human-mobility networks, country income, and labor productivity. *Network Science*, 3(3):377–407, 2015.

[83] C. Faloutsos, K. S. McCurley, and A. Tomkins. Fast discovery of connection subgraphs. In *KDD*, 2004.

[84] Y. Fang, R. Cheng, Y. Chen, S. Luo, and J. Hu. Effective and efficient attributed community search. *The VLDB Journal*, 26(6):803–828, 2017.

[85] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu. Effective community search over large spatial graphs. *Proceedings of the VLDB Endowment*, 10(6):709–720, 2017.

[86] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin. A survey of community search over big graphs. *arXiv preprint arXiv:1904.12539*, 2019.

[87] U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. *Algorithmica*, 29(3), 2001.

[88] L. Feldman, T. A. Myers, J. D. Hmielowski, and A. Leiserowitz. The mutual reinforcement of media selectivity and effects: Testing the reinforcing spirals framework in the context of global warming. *Journal of Communication*, 64(4):590–611, 2014.

[89] S. Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[90] C. Franzoni, G. Scellato, and P. Stephan. Foreign-born scientists: mobility patterns for 16 countries. *Nature Biotechnology*, 30(12):1250, 2012.

[91] E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou. MotifCut: regulatory motifs finding with maximum density subgraphs. In *ISMB*, 2006.

[92] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[93] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[94] E. Galbrun, A. Gionis, and N. Tatti. Top-k overlapping densest subgraphs. *Data Mining and Knowledge Discovery*, 30(5):1134–1165, 2016.

[95] E. Galimberti, A. Barrat, F. Bonchi, C. Cattuto, and F. Gullo. Mining (maximal) span-cores from temporal networks. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 107–116. ACM, 2018.

[96] E. Galimberti, F. Bonchi, and F. Gullo. Core decomposition and densest subgraph in multilayer networks. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1807–1816. ACM, 2017.

[97] E. Galimberti, F. Bonchi, F. Gullo, and T. Lanciano. Core decomposition in multilayer networks: Theory, algorithms, and applications. *arXiv preprint arXiv:1812.08712*, 2018.

[98] E. Galimberti, M. Ciaperoni, A. Barrat, F. Bonchi, C. Cattuto, and F. Gullo. Span-core decomposition for temporal networks: Algorithms and applications. *arXiv preprint arXiv:1910.03645*, 2019.

[99] E. Galimberti, C. Madeddu, F. Bonchi, and G. Ruffo. Visualizing structural balance in signed networks. In *International Conference on Complex Networks and their Applications*. Springer, 2019.

[100] M. Gao, E.-P. Lim, D. Lo, and P. K. Prasetyo. On detecting maximal quasi antagonistic communities in signed graphs. *Data mining and knowledge discovery*, 30(1):99–146, 2016.

[101] A. Garas, F. Schweitzer, and S. Havlin. A k -shell decomposition method for weighted networks. *New Journal of Physics*, 14(8):083030, 2012.

[102] D. Garcia, P. Mavrodiev, and F. Schweitzer. Social resilience in online communities: The autopsy of friendster. *CoRR*, abs/1302.6109, 2013.

[103] K. Garimella, G. De Francisci Morales, A. Gionis, and M. Mathioudakis. Reducing controversy by connecting opposing views. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 81–90, 2017.

[104] K. Garrett and N. J. Stroud. Partisan paths to exposure diversity: Differences in pro-and counter-attitudinal news consumption. *Journal of Communication*, 64(4):680–701, 2014.

[105] L. Gauvin, A. Panisson, A. Barrat, and C. Cattuto. Revealing latent factors of temporal networks for mesoscale intervention in epidemic spread. *arXiv preprint arXiv:1501.02758*, 2015.

[106] L. Gauvin, A. Panisson, and C. Cattuto. Detecting the community structure and activity patterns of temporal networks: a non-negative tensor factorization approach. *PLOS ONE*, 9(1):e86028, 2014.

[107] V. Gemmetto, A. Barrat, and C. Cattuto. Mitigation of infectious disease at school: targeted class closure vs school closure. *BMC infectious diseases*, 14(1):695, Dec. 2014.

[108] A. Geuna. *Global mobility of research scientists: The economics of who goes where and why.* Academic Press, Cambridge, Massachusetts, 2015.

[109] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis. D-cores: measuring collaboration of directed graphs based on degeneracy. *Knowledge and information systems*, 35(2):311–343, 2013.

[110] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, 2005.

[111] C. Gini. Variabilità e mutabilità. *Reprinted in Memorie di metodologica statistica (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi*, 1912.

[112] I. Giotis and V. Guruswami. Correlation clustering with a fixed number of clusters. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1167–1176, 2006.

[113] L. M. Glass and R. J. Glass. Social contact networks for the spread of pandemic influenza in children and teenagers. *BMC public health*, 8(1):61, 2008.

[114] A. V. Goldberg. Finding a maximum density subgraph. Technical report, University of California at Berkeley, 1984.

[115] P. Goyal and E. Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78 – 94, 2018.

[116] E. Graells-Garrido, M. Lalmas, and D. Quercia. People of opposing views can share common interests. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 281–282, 2014.

[117] G. Grimmett. *What is Percolation?* Springer, 1999.

[118] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[119] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In *Proceedings of the 13th international conference on World Wide Web*, pages 403–412, 2004.

[120] P. Hage, F. Harary, and F. Harary. Structural models in anthropology: Cambridge studies in social anthropology. 1983.

[121] F. Harary. On the notion of balance of a signed graph. *The Michigan Mathematical Journal*, 2(2):143–146, 1953.

[122] F. Harary and J. A. Kabell. A simple algorithm to detect balance in signed graphs. *Mathematical Social Sciences*, 1(1):131–136, 1980.

[123] J. Healy, J. Janssen, E. Milios, and W. Aiello. Characterization of graphs using degree cores. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 137–148. Springer, 2006.

[124] F. Heider. *The psychology of interpersonal relations.* Psychology Press, 2013.

[125] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on visualization and computer graphics*, 6(1):24–43, 2000.

[126] A.-S. Himmel, H. Molter, R. Niedermeier, and M. Sorge. Enumerating maximal cliques in temporal graphs. In *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*, pages 337–344. IEEE, 2016.

[127] P. Holme and J. Saramäki. Temporal networks. *Physics reports*, 519(3):97–125, 2012.

[128] Y. Hou, J. Li, and Y. Pan. On the Laplacian eigenvalues of signed graphs. *Linear and Multilinear Algebra*, 51(1):21–30, 2003.

[129] Y. P. Hou. Bounds for the least Laplacian eigenvalue of a signed graph. *Acta Mathematica Sinica*, 21(4):955–960, 2005.

[130] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[131] X. Huang and L. V. Lakshmanan. Attribute-driven community search. *Proceedings of the VLDB Endowment*, 10(9):949–960, 2017.

[132] X. Huang, L. V. S. Lakshmanan, and J. Xu. Community search over big graphs: Models, algorithms, and opportunities. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*, pages 1451–1454, 2017.

[133] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Pocket switched networks and human mobility in conference environments. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 244–251. ACM, 2005.

[134] A. Inokuchi and T. Washio. Mining frequent graph sequence patterns induced by vertices. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 466–477. SIAM, 2010.

[135] R. Interdonato, A. Tagarelli, D. Ienco, A. Sallaberry, and P. Poncelet. Local community detection in multilayer networks. *Data Min. Knowl. Discov.*, 31(5):1444–1479, 2017.

[136] V. Jethava and N. Beerenwinkel. Finding dense subgraphs in relational graphs. In *ECML-PKDD*, pages 641–654, 2015.

[137] D. Jiang and J. Pei. Mining frequent cross-graph quasi-cliques. *TKDD*, 2(4):16, 2009.

[138] T. Kamada, S. Kawai, et al. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989.

[139] L. Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[140] M. Kaufmann and D. Wagner. *Drawing graphs: methods and models*, volume 2025. Springer, 2003.

[141] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse. Identification of influential spreaders in complex networks. *Nature physics*, 6(11):888, 2010.

[142] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter. Multilayer networks. *Journal of complex networks*, 2(3):203–271, 2014.

[143] J. M. Kleinberg. Hubs, authorities, and communities. *ACM computing surveys (CSUR)*, 31(4es):5, 1999.

[144] J. Kleinnijenhuis and W. De Nooy. Adjustment of issue positions based on network strategies in an election campaign: A two-mode network autoregression model with cross-nested random effects. *Social Networks*, 35(2):168–177, 2013.

[145] I. M. Kloumann and J. M. Kleinberg. Community membership identification from small seed sets. In *KDD*, 2014.

[146] J. Klugman. Human development report 2009. overcoming barriers: Human mobility and development. 2009.

[147] Y. Koren, L. Carmel, and D. Harel. Ace: A fast multiscale eigenvectors computation for drawing huge graphs. In *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.*, pages 137–144. IEEE, 2002.

[148] G. Kortsarz and D. Peleg. Generating sparse 2-spanners. *Journal of Algorithms*, 17(2):222–236, 1994.

[149] L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki. Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics*, page P11005, 2011.

[150] M. Krzywinski, I. Birol, S. J. Jones, and M. A. Marra. Hive plots—rational approach to visualizing networks. *Briefings in bioinformatics*, 13(5):627–644, 2011.

[151] J. Kunegis, A. Lommatzsch, and C. Bauckhage. The slashdot zoo: mining a social network with negative edges. In *Proceedings of the 18th international conference on World wide web*, pages 741–750, 2009.

[152] J. Kunegis, S. Schmidt, A. Lommatzsch, J. Lerner, E. W. De Luca, and S. Albayrak. Spectral analysis of signed graphs for clustering, prediction and visualization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 559–570, 2010.

[153] M. Lai, V. Patti, G. Ruffo, and P. Rosso. Stance evolution and twitter interactions in an italian political debate. In *International Conference on Applications of Natural Language to Information Systems*, pages 15–27, 2018.

[154] M. A. Langston and et al. A combinatorial approach to the analysis of differential gene expression data: The use of graph algorithms for disease prediction and screening. In *Methods of Microarray Data Analysis IV*. 2005.

[155] K.-M. Lee, B. Min, and K.-I. Goh. Towards real-world complexity: an introduction to multiplex networks. *The European Physical Journal B*, 88(2), 2015.

[156] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336. Springer, 2010.

[157] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World wide web*, pages 641–650, 2010.

[158] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1361–1370, 2010.

[159] C. W.-k. Leung, E.-P. Lim, D. Lo, and J. Weng. Mining interesting link formation rules in social networks. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 209–218. ACM, 2010.

[160] R.-H. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai. Persistent community search in temporal networks. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 797–808. IEEE, 2018.

[161] R.-H. Li, J. X. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. *IEEE Transactions on Knowledge and Data Engineering*, 26(10):2453–2465, 2014.

[162] Y. Li, W. Chen, Y. Wang, and Z.-L. Zhang. Influence diffusion dynamics and influence maximization in social networks with friend and foe relationships. In *Proceedings of the sixth ACM international conference on Web search and data mining (WSDM)*, pages 657–666, 2013.

[163] Q. V. Liao and W.-T. Fu. Can you hear me now?: mitigating the echo chamber effect by source position indicators. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pages 184–196, 2014.

[164] Q. V. Liao and W.-T. Fu. Expert voices in echo chambers: effects of source expertise indicators on exposure to diverse opinions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2745–2754, 2014.

[165] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou. Efficient (a, β)-core computation: an index-based approach. In *The World Wide Web Conference*, pages 1130–1141. ACM, 2019.

[166] P. Liu, A. R. Benson, and M. Charikar. Sampling methods for counting temporal motifs. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 294–302. ACM, 2019.

[167] D. Lo, D. Surian, K. Zhang, and E.-P. Lim. Mining direct antagonistic communities in explicit trust networks. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1013–1018. ACM, 2011.

[168] R. D. Luce. Connectivity and generalized cliques in sociometric group structure. *Psychometrika*, 15(2):169–190, 1950.

[169] H. Ma, M. R. Lyu, and I. King. Learning to recommend with trust and distrust relationships. In *Proceedings of the 3rd ACM conference on Recommender systems*, pages 189–196, 2009.

[170] S. Ma, R. Hu, L. Wang, X. Lin, and J. Huai. Fast computation of dense temporal subgraphs. In *ICDE*, pages 361–372, 2017.

[171] F. Malliaros, C. Giatsidis, A. Papadopoulos, and M. Vazirgiannis. The core decomposition of networks: Theory, algorithms and applications. 2019.

[172] S. A. Marvel, S. H. Strogatz, and J. M. Kleinberg. The energy landscape of social balance. *Physical Review Letters*, 103(19), 2009.

[173] S. Maslov and K. Sneppen. Specificity and stability in topology of protein networks. *Science*, 296(5569):910–913, 2002.

[174] R. Mastrandrea, J. Fournet, and A. Barrat. Contact patterns in a high school: A comparison between data collected using wearable sensors, contact diaries and friendship surveys. *PLoS ONE*, 10(9):1–26, 09 2015.

[175] H. Matsuda, T. Ishihara, and A. Hashimoto. Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theoretical Computer Science*, 210(2):305–325, 1999.

[176] R. Mikolajczyk, M. Akmatov, S. Rastin, and M. Kretzschmar. Social contacts of school children and the transmission of respiratory-spread pathogens. *Epidemiology & Infection*, 136(6):813–822, 2008.

[177] H. F. Moed, A. Plume, et al. Studying scientific migration in scopus. *Scientometrics*, 94(3):929–942, 2013.

[178] R. J. Mokken. Cliques, clubs and clans. *Quality & Quantity*, 13(2):161–173, 1979.

[179] M. Mongiovi, P. Bogdanov, R. Ranca, E. E. Papalexakis, C. Faloutsos, and A. K. Singh. Netspot: Spotting significant anomalous regions on dynamic networks. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 28–36. SIAM, 2013.

[180] A. Montresor, F. De Pellegrini, and D. Miorandi. Distributed k-core decomposition. *IEEE Transactions on parallel and distributed systems*, 24(2):288–300, 2013.

[181] P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, and J.-P. Onnela. Community structure in time-dependent, multiscale, and multiplex networks. *science*, 328(5980):876–878, 2010.

[182] S. A. Munson, S. Y. Lee, and P. Resnick. Encouraging reading of diverse political viewpoints with a browser widget. In *ICWSM*, 2013.

[183] M. A. U. Nasir, A. Gionis, G. D. F. Morales, and S. Girdzijauskas. Fully dynamic algorithm for top-k densest subgraphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1817–1826. ACM, 2017.

[184] M. Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[185] OECD. *A profile of immigrant populations in the 21st century: data from OECD countries*. OECD Paris, Paris, France, 2008.

[186] B. Ordozgoiti and A. Gionis. Reconciliation k-median: Clustering with non-polarized representatives. In *The World Wide Web Conference*, pages 1387–1397. ACM, 2019.

[187] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[188] E. E. Papalexakis, L. Akoglu, and D. Ience. Do more views of a graph help? community detection and clustering in multi-graphs. In *FUSION*, pages 899–905, 2013.

[189] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[190] N. Perra and S. Fortunato. Spectral centrality measures in complex networks. *Physical Review E*, 78(3):036107, 2008.

[191] T. Pisanski and J. Shawe-Taylor. Characterizing graph drawing with eigenvectors. *Journal of Chemical Information and Computer Sciences*, 40(3):567–571, 2000.

[192] A. V. Proskurnikov, A. S. Matveev, and M. Cao. Opinion dynamics in social networks with hostile camps: Consensus vs. polarization. *IEEE Transactions on Automatic Control*, 61(6):1524–1536, 2015.

[193] E. Pugliese, G. Cimini, A. Patelli, A. Zaccaria, L. Pietronero, and A. Gabrielli. Unfolding the innovation system for the development of countries: co-evolution of science, technology and production. *arXiv preprint arXiv:1707.05146*, 2017.

[194] S. Redner. Social balance on networks: The dynamics of friendship and hatred. In *APS Meeting Abstracts*, 2006.

[195] A. Reinthal, A. Andersson, E. Norlander, P. Stålhammar, S. Norlin, et al. Finding the densest common subgraph with linear programming. 2016.

[196] S. Rinzivillo, S. Mainardi, F. Pezzoni, M. Coscia, D. Pedreschi, and F. Giannotti. Discovering the geographical borders of human mobility. *KI-Künstliche Intelligenz*, 26(3):253–260, 2012.

[197] C. Robinson and B. Dilkina. A machine learning approach to modeling human migration. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, page 30. ACM, 2018.

[198] A. Roli. An introduction to complex system science.

[199] P. Rozenshtein, F. Bonchi, A. Gionis, M. Sozio, and N. Tatti. Finding events in temporal networks: Segmentation meets densest-subgraph discovery. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 397–406. IEEE, 2018.

[200] P. Rozenshtein, N. Tatti, and A. Gionis. Finding dynamic dense subgraphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 11(3):27, 2017.

[201] N. Ruchansky, F. Bonchi, D. García-Soriano, F. Gullo, and N. Kourtellis. To be connected, or not to be connected: That is the minimum inefficiency subgraph problem. In *CIKM 2017*.

[202] N. Ruchansky, F. Bonchi, D. García-Soriano, F. Gullo, and N. Kourtellis. The minimum wiener connector problem. In *SIGMOD*, 2015.

[203] M. Salathé, M. Kazandjieva, J. W. Lee, P. Levis, M. W. Feldman, and J. H. Jones. A high-resolution human contact network for infectious disease transmission. *Proceedings of the National Academy of Sciences*, 107(51):22020–22025, 2010.

[204] A. Sapienza, A. Panisson, J. Wu, L. Gauvin, and C. Cattuto. Detecting anomalies in time-varying networks using tensor decomposition. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*, pages 516–523. IEEE, 2015.

[205] A. E. Saríyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and Ü. V. Çatalyürek. Streaming algorithms for k-core decomposition. *Proceedings of the VLDB Endowment*, 6(6):433–444, 2013.

[206] A. Saxenian. From brain drain to brain circulation: Transnational communities and regional upgrading in india and china. *Studies in comparative international development*, 40(2):35–61, 2005.

[207] F. Schiantarelli. Global economic prospects 2006: economic implications of remittances and migration. *The World Bank*, 2005.

[208] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[209] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept*. *Journal of Mathematical sociology*, 6(1):139–154, 1978.

[210] K. Semertzidis, E. Pitoura, E. Terzi, and P. Tsaparas. Best friends forever (bff): Finding lasting dense subgraphs. *arXiv preprint arXiv:1612.05440*, 2016.

[211] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

[212] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003.

[213] C.-Y. Shen, H.-H. Shuai, D.-N. Yang, Y.-F. Lan, W.-C. Lee, P. S. Yu, and M.-S. Chen. Forming online support groups for internet and behavior related addictions. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 163–172. ACM, 2015.

[214] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[215] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *KDD*, pages 939–948, 2010.

[216] J. Stehlé, F. Charbonnier, T. Picard, C. Cattuto, and A. Barrat. Gender homophily from spatial behavior in a primary school: A sociometric study. *Social Networks*, 35:604–613, 2013.

[217] J. Stehlé, N. Voirin, A. Barrat, C. Cattuto, L. Isella, J.-F. Pinton, M. Quaggiotto, W. Van den Broeck, C. Régis, B. Lina, and P. Vanhems. High-resolution measurements of face-to-face contact patterns in a primary school. *PLoS ONE*, 6(8):e23176, 08 2011.

[218] C. Swamy. Correlation clustering: maximizing agreements via semidefinite programming. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 526–527, 2004.

[219] P. Symeonidis, E. Tiakas, and Y. Manolopoulos. Transitive node similarity for link prediction in social networks with positive and negative links. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 183–190, 2010.

[220] A. Tagarelli, A. Amelio, and F. Gullo. Ensemble-based community detection in multilayer networks. *Data Mining and Knowledge Discovery (DAMI)*, 31(5):1506–1543, 2017.

[221] J. Tang, C. Aggarwal, and H. Liu. Node classification in signed social networks. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 54–62, 2016.

[222] J. Tang, C. Aggarwal, and H. Liu. Recommendations in signed social networks. In *Proceedings of the 25th International Conference on World Wide Web*, pages 31–40, 2016.

[223] J. Tang, Y. Chang, C. Aggarwal, and H. Liu. A survey of signed network mining in social media. *ACM Computing Surveys (CSUR)*, 49(3):42, 2016.

[224] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[225] L. Tang, X. Wang, and H. Liu. Community detection in multi-dimensional networks. Technical report, DTIC Document, 2010.

[226] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, pages 404–413, 2006.

[227] C. Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th international conference on world wide web*, pages 1122–1132. International World Wide Web Conferences Steering Committee, 2015.

[228] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 104–112. ACM, 2013.

[229] A. Urbinati, E. Galimberti, and G. Ruffo. Hubs and authorities of the scientific migration network. *arXiv preprint arXiv:1907.07175*, 2019.

[230] L. Verginer and M. Riccaboni. Brain-circulation network: The global mobility of the life scientists. 2018.

[231] T. Viard, M. Latapy, and C. Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, 2016.

[232] P. Victor, C. Cornelis, M. De Cock, and A. M. Teredesai. Trust-and distrust-based recommendations for controversial reviews. *IEEE Intelligent Systems*, 26(1):48–55, 2011.

[233] V. Vydiswaran, C. Zhai, D. Roth, and P. Pirolli. Overcoming bias to learn about controversial topics. *Journal of the Association for Information Science and Technology*, 66(8):1655–1672, 2015.

[234] N. Wang, J. Zhang, K.-L. Tan, and A. K. Tung. On triangulation-based dense neighborhood graph discovery. *Proceedings of the VLDB Endowment*, 4(2):58–68, 2010.

[235] S. Wasserman, K. Faust, et al. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[236] M. Wojcieszak and D. Mutz. Online groups and political discourse: Do online discussion spaces facilitate exposure to political disagreement? *Journal of communication*, 59(1):40–56, 2009.

[237] H. Wu, J. Cheng, Y. Lu, Y. Ke, Y. Huang, D. Yan, and H. Wu. Core decomposition in large temporal graphs. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 649–658. IEEE, 2015.

[238] Y. Wu, R. Jin, X. Zhu, and X. Zhang. Finding dense and connected subgraphs in dual networks. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 915–926. IEEE, 2015.

[239] S. Wuchty and E. Almaas. Peeling the yeast protein network. *Proteomics*, 5(2):444–449, 2005.

[240] W. Xia, M. Cao, and K. H. Johansson. Structural balance and opinion separation in trust–mistrust social networks. *IEEE Transactions on Control of Network Systems*, 3(1):46–56, 2015.

[241] X. Yan, X. Zhou, and J. Han. Mining closed relational graphs with connectivity constraints. In *KDD*, pages 324–333, 2005.

[242] Y. Yan, Y. Bian, D. Luo, D. Lee, and X. Zhang. Constrained local graph clustering by colored random walk. In *The World Wide Web Conference*, pages 2137–2146. ACM, 2019.

[243] Z. M. Yin and S. S. Khaing. Multi-layered graph clustering in finding the community cores. *Int. J. Adv. Res. Comput. Eng. Technol*, 2, 2013.

[244] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

[245] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. When engagement meets similarity: efficient (k, r)-core computation on social networks. *Proceedings of the VLDB Endowment*, 10(10):998–1009, 2017.

[246] H. Zhang, H. Zhao, W. Cai, J. Liu, and W. Zhou. Using the k-core decomposition to analyze the static structure of large-scale software systems. *J. Supercomputing*, 53(2), 2010.

[247] R. Zhu, Z. Zou, and J. Li. Diversified coherent core search on multi-layer graphs. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 701–712. IEEE, 2018.

[248] P. Zschoche, T. Fluschnik, H. Molter, and R. Niedermeier. The complexity of finding small separators in temporal graphs. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.