UNIVERSITÀ DEGLI STUDI DI CATANIA

DIPARTIMENTO DI INGEGNERIA ELETTRICA ELETTRONICA E INFORMATICA

DOTTORATO DI RICERCA IN INGEGNERIA DEI SISTEMI ENERGETICA, INFORMATICA E DELLE TELECOMUNICAZIONI

Learning NP-hard problems on networks using Geometric Deep Learning

Ph.D. Thesis

Ph.D. Candidate:
Marco Grassia

Advisor:
Prof. Giuseppe Mangioni

XXXIV CYCLE

There is a theory which states that if ever anyone discovers exactly what the Universe is for and why it is here, it will instantly disappear and be replaced by something even more bizarre and inexplicable.
There is another theory which states that this has already happened.
— Douglas Adams, *The Restaurant at the End of the Universe*

To my parents,
who encouraged me to take the leap and supported me through these years.

Acknowledgements

I would first like to express my sincere gratitude to my supervisor, Professor Giuseppe Mangioni, for his invaluable advice and feedback, for the inspiring ideas and stimulating discussions, for his support, and, maybe most importantly, for his perseverance.

I would like to thank Professor Paolo Pietro Arena, Ph.D. Coordinator, for his guidance during my Ph.D.

I would also like to thank Professor Vincenza Carchiolo and Professor Michele Malgeri for their support.

I would like to offer my special thanks to Dr. Manlio De Domenico for his advice and for the interesting research questions.

I would like to thank Dr. Carlo Cannistraci for the internship opportunity in his lab.

I would like to extend my sincere thanks to Dr. Alessandra Sala, my former manager during my Bell Labs internship, for believing in me and for her precious advice that helped me during my Ph.D.

I am grateful to my parents, *mamma* Giovanna e *papà* Enzo, for everything they have done for me to help me achieve this goal, to my brother Stefano and to my grandma, *nonna* Maria, for supporting me and for cheering after every achievement. You are always there for me. Thank you.

Last but not least, I would also deeply thank Alessia, Egle and Luigi for all the time spent together, for all the hugs and for all the happy distractions. You endured this long path with me, always offering support and love, and made the hard moments easier.

Abstract

Many systems from a wide range of domains can be modeled as networks, i.e., a set of nodes that represent the entities of the system and a set of links between nodes that represent relations among them. For instance, social networks can be used to describe how people interact with each other, computer networks can represent communication channels (physical mediums, logical connections, etc.), trophic networks show how animals feed, and so on. The underlying complexity of the modeled systems translates into non-trivial patterns in the networks, and, thus, they are often called *complex networks*. The increasing importance of networks in many domains (e.g., social, technological, biological, etc.) and the need of methodologies and tools to study complex networks led to the rise of *Network Science*, the field that studies networks and their applications. However, many problems on networks remain hard to define formally and even harder to solve exactly (for instance for computational constraints), requiring sub-optimal heuristics. In this work we show how Geometric Deep Learning, the generalization of Deep Learning to non-Euclidean domains like graphs, can aid the computation of NP-hard problems and learn heuristics from the data. Specifically, we define a framework, namely *GDM*, to learn how to solve the *Network Dismantling* and *Link Building* problems on the optimal solutions computed on small synthetic graphs, and then generalize to large real-world networks. For both problems, the state-of-the-art heuristics are outperformed significantly. This result can be achieved thanks to the generalization performance of the Graph Neural Network (GNN) layers employed. We also define *mGNN*, a framework to generalize any GNN to *Multilayer Networks*, and *wsGAT*, a new GNN extending the Graph Attention Network (GAT) layers to handle signed and weighted networks.

Publications

In this section are listed the publications made during my three-year Ph.D. program.
The ones covered in this thesis are marked with a *.
Some publications are still under review and reported as "Submitted to ...".

* Grassia, M., De Domenico, M. & Mangioni, G. Machine learning dismantling and early-warning signals of disintegration in complex systems. *Nature Communications* **12**, 5190 (2021). URL https://doi.org/10.1038/s41467-021-25485-8

* Grassia, M. & Mangioni, G. Weighted and signed graph attention networks. *Submitted to the 10th International Conference on Complex Networks and their Applications (COMPLEX NETWORKS 2021)* (2021)

* Grassia, M., De Domenico, M. & Mangioni, G. mGNN: Generalizing the Graph Neural Networks to the Multilayer Case. *Submitted to IEEE Transactions on Neural Networks and Learning Systems* (2021)

* Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Efficient node rank improvement via link building using geometric deep learning. *Submitted to IEEE Transactions on Knowledge and Data Engineering* (2021)

Grassia, M., Mangioni, G., Schiavo, S. & Traverso, S. International food trade and vulnerability to shocks: insights from network-based simulations. *Submitted to Environmental Research Letters* (2021)

Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Food recommendation in a worksite canteen. In *COMPLEXIS*, 117–124 (2021)

Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. A network-based analysis of a worksite canteen dataset. *Big Data and Cognitive Computing* **5**, 11 (2021)

Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Analysis of the co-authorship sub-networks of italian academic researchers. *Submitted to the 2021 IEEE/ACM International Conference on Advances in Social Network Analysis and Mining (ASONAM)* (2021)

Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Preliminary characterization of italian academic scholars by their bibliometrics. *Submitted to 14th International Symposium on Intelligent Distributed Computing (IDC 2021)* (2021)

Grassia, M., Mangioni, G., Schiavo, S. & Traverso, S. (unintended) consequences of export restrictions on medical goods during the covid-19 pandemic. *arXiv preprint arXiv:2007.11941* (2020)

Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Group cohesion assessment in networks. In *Complex Networks XI*, 16–25 (Springer, Cham, 2020)

Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. A network-based analysis to understand food-habits of a multi-company canteen's customers. In *Proceedings of the 22nd International Conference on Information Integration and Web-based Applications & Services*, 352–356 (2020)

Lauri, J., Dutta, S., Grassia, M. & Ajwani, D. Learning fine-grained search space pruning and heuristics for combinatorial optimization (2020). 2001.01230

Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Network robustness improvement via long-range links. *Computational Social Networks* **6**, 1–16 (2019)

Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. A pagerank inspired approach to measure network cohesiveness. In *International Conference on Internet and Distributed Computing Systems*, 349–356 (Springer, Cham, 2019)

Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Strategies comparison in link building problem. In *International Symposium on Intelligent and Distributed Computing*, 197–202 (Springer, Cham, 2019)

Grassia, M., Lauri, J., Dutta, S. & Ajwani, D. Learning multi-stage sparsification for maximum clique enumeration. *arXiv preprint arXiv:1910.00517* (2019)

Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Exploiting long distance connections to strengthen network robustness. In *International Conference on Internet and Distributed Computing Systems*, 270–277 (Springer, Cham, 2018)

Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Long distance in-links for ranking enhancement. In *International Symposium on Intelligent and Distributed Computing*, 3–10 (Springer, Cham, 2018)

Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Climbing ranking position via long-distance backlinks. In *International Conference on Internet and Distributed Computing Systems*, 100–108 (Springer, Cham, 2018)

Table of contents

Chapter 1

Introduction to Network Science and Machine Learning

In this Chapter we introduce the motivations and some fundamentals of Network Science and of Machine Learning, the two fields at the core of this thesis, and that should help the Reader understand Geometric Deep Learning and the problems addressed in the following chapters.

1.1 Network Science

In the "century of complexity" [21] and in the time of the new challenges offered by the large availability of real-world data and by computational capabilities of the modern hardware, there is a growing interest in the study of complex systems — i.e., systems with many interacting entities — and in the study of their *emergent properties* — i.e., those properties of the system that its individual components do not have —. We could say that complex systems are everywhere: examples are social networks, the Internet network, cities, trophic relations, the human body, etc. While such systems are widely represented as networks (graphs) [22], hence

Fig. 1.1 Example of a technological complex network.

the name *complex networks*, graph theory alone does not provide the right tools to study them, as its main focus is on "providing rigorous proofs for graph properties" [23], often on tractable graphs like random or smaller ones. In this scenario, *Network Science* was born in the recent past as an interdisciplinary attempt of applying tools from various fields like mathematics, physics, statistics and computer science, to study the network representations of real-world complex systems with the final goal being understanding them and their emergent properties, either structural or dynamics-related. However, it should be noted that Network Science has quite a long history, and that it has strong roots in social psychology, sociology, and anthropology [24]. In the past, there have been many attempts to formally define Network Science, one of the first was made in 2005 by the U.S. National Research Council [25], which defined it as "the study of network representations of physical, biological, and social phenomena leading to predictive models of these phenomena".

In the following, we introduce the Network Science concepts and terminology needed to understand the problems addressed in this thesis. Specifically, after an introduction to the concept of network and to some network types, we give an overlook of some typical analysis tools used, then we introduce some problems on networks, and multilayer networks. For more details, we refer the Reader to other works and surveys [26, 27].

1.1.1 Networks

Formally, a network can be represented as a graph $\mathscr{G} = (V, E)$, where V is the set of the *nodes* (or *vertices*), the components of the system, and E us the set of *links* (or *edges*) that represent interactions between them.

The generic definition of nodes and links allows a range of applicability so wide that networks are often defined ubiquitous: nodes can be computers, people, cities, proteins, animal species, neurons, machines, etc. and links can encode social relationships, communication channels, regulatory interactions, predator-prey interactions, synaptic connections, events, etc. Just to mention a few applications, networks have been employed in biology [28], ecology [29], neuroscience [30], economy [31], finance [32], engineering [33], physics [34], social [35, 36] and computer science [37]. In fact, one of the advantages of the network modelling is that the networks are independent of the domain of the system. That is, networks are domain-agnostic [38] and two systems from very different domains can have similar or even the same network representation. This allows researchers to use the same Network Science tools to solve problems in any domain, even without domain-specific knowledge.

1.1.1.1 Different types of networks

To model a wide variety of systems and data, various types of networks have been defined. In the following, we mention the most important ones.

Directed networks Links can be *directed* [39], meaning that the interaction $e_{i,j}$ between two nodes u and v is asymmetric as it originates from node i and targets node j. On the other hand, in *undirected* networks a link has no direction and $e_{ij} = e_{ji}$.

Weighted networks Each link from node i to node j in the network could have a different *weight* [40], denoted with w_{ij}. In this case, the network is said to be *weighted*. On the other hand, in *unweighted* (or *binary*) networks we have that $w_{ij} = 1 \ \ \forall i, j$. The meaning of link weights depends on the system. For instance, weights can represent the strength of a relationship, the costs of an interaction, the estimated trip time, the distance between the two nodes, etc.

Signed networks *Signed networks* [41] are sub-type of weighted networks where link weights w_{ij} can also be negative. This is the case, for instance, of correlation networks where each link is the correlation among the nodes [42, 43], or the case of trust networks [44–46] where trust scores between users can also be negative (e.g., to represent mistrust).

Feature-rich networks Networks can also come with some kind of information associated to the entire graph, to the nodes or to the links. For instance, in social networks nodes may include personal information like birthday date, hair color, height, etc., and links can include the date two nodes first met, and so on.

1.1.1.2 Adjacency matrix

A common way to represent a network is though its *adjacency matrix* [47], usually denoted with $\mathbf{A}$, a square matrix of order N, where N is the number of nodes in the network. Each element of $\mathbf{A}$, a_{ij}, is either 1 if there is a [directed] link from node i to j or 0 otherwise. Note that $a_{ii} = 1$ means that node i has a *self-loop*, i.e., an incoming link from itself. The meaning of self-loops depends on the modelled system.

1.1.2 Network analysis

Due to the large size of real-world networks, a thorough analysis is not feasible. Instead, networks are often studied from a statistical perspective (e.g., by looking at degree and

strength distribution, assortativity, clustering coefficients, etc.) and by looking at some other properties that are meaningful for the specific domain. In the following, we introduce some of these tools and properties.

1.1.2.1 Structural analysis

Degree, average degree and degree distribution The *degree* k_i of a node i is the number of its links.

$$k_i = \sum_j^N a_{ij} = \sum_j^N a_{ji} \tag{1.1}$$

In directed networks, we distinguish between the number of incoming links (*in-links*), called *in-degree* k_i^{in}, and the number of outgoing links (*out-links*), called *out-degree* k_i^{out}.

$$k_i^{\text{in}} = \sum_j^N a_{ji} \tag{1.2}$$

$$k_i^{\text{out}} = \sum_j^N a_{ij} \tag{1.3}$$

$$\tag{1.4}$$

Of course, it follows that the (total) degree is $k_i^{\text{total}} = k_i^{\text{in}} + k_i^{\text{out}}$. The *average [in|out] degree* of a network, usually denoted as $\langle k \rangle$, is the average [in|out] degree of the network's nodes. However, the average value does not tell much about the degree. For this reason, it is common to visualize and analyze the *degree distribution* $P(k)$ of the network, i.e., the normalized probability that a node has degree k.

Strength, average strength and strength distribution In the case of weighted networks, another interesting measure is the *strength* of the nodes, i.e., the sum of the weight of its links.

$$s_i = \sum_j^N a_{ij} w_{ij} = \sum_j^N a_{ji} w_{ji} \tag{1.5}$$

Like in the case of the degree, if the network is also directed, one can compute the in-strength and the out-strength, i.e., the strength of the incoming and outgoing links of that node.

$$s_i^{\text{in}} = \sum_j^N a_{ji} w_{ji} \tag{1.6}$$

$$s_i^{\text{out}} = \sum_j^N a_{ij} w_{ij} \tag{1.7}$$

$$\tag{1.8}$$

The total strength can be computed as $s_i^{\text{total}} = s_i^{\text{in}} + s_i^{\text{out}}$. Again, one can also consider the average value $\langle s \rangle$, and the strength distribution $P(s)$.

Network diameter The [directed] *diameter* of a network is the length of the longest shortest path between any two nodes in the network.

Link density and reciprocity The *density* of links is the ratio of links in the network to the theoretically possible maximum.

The *reciprocity* of links in a directed network is the ratio of links reciprocated in the other direction. That is, a link e_{ij} is reciprocated if the link e_{ji} also exists in the network.

Clustering coefficients The *local clustering coefficient* C_i is a local measure of the cliquishness of the neighborhood of node i [48], used to describe the 'small-world' effect. That is, it quantifies the probability that two of i's neighbors are also connected (i.e., the local ratio of closed triplets that form a triangle, where a triplet is three nodes that connected by two or three links). For an undirected graph, it can be computed as [48]

$$C_i = \frac{2|\{e_{jk} : n_j, n_k \in \mathcal{N}_i, e_{jk} \in E\}|}{k_i(k_i - 1)} \tag{1.9}$$

where $\mathcal{N}_i$ is the neighborhood of node i, n_j and n_k are its neighbors and e_{jk} is the edge between them, and k_i is the degree of node i. Note that this measure is defined between $0 \leq C_i \leq 1$ as $\frac{k_i(k_i-1)}{2}$ is the maximum number of possible links between the neighbors of node i. The local clustering coefficient can also be computed for directed and/or weighted graphs, but we do not cover these extensions here.

It is also common to consider the *average clustering coefficient* C, i.e., the average C_i value for a network:

$$C = \frac{1}{N} \sum_i^N C_i \tag{1.10}$$

The *global clustering coefficient T* [22], also known as *transitivity*, is another measure of clustering, however it is a global defined for the entire network as:

$$T = 3\,\frac{\text{number of triangles}}{\text{number of triplets}} \tag{1.11}$$

Assortativity The *assortativity* coefficient r of a network quantifies the tendency of nodes to connect with nodes with similar characteristics [49]. r is a value between $-1 \leq r \leq 1$ defined as [49]

$$r = \frac{\sum_i e_{ii} - \sum_i a_i b_i}{1 - \sum_i a_i b_i} \tag{1.12}$$

where $a_i = \sum_j e_{ij}$ and $b_i = \sum_i e_{ij}$, and e_{ij} is the fraction of links that connect nodes of type i to nodes of type j. Note that $a_i = b_i$ in undirected networks and that $\sum_{ij} e_{ij} = 1$. If $r = 0$, we say that there is no assortativity and if $r < 0$ we talk about disassortativity.

However, it is common to compute the *scalar assortativity* of the node's degree to measure if nodes preferentially connect to nodes with similar degree. The *scalar assortativity* is defined similarly as before, the main difference being that it does not consider node types but values associated to the nodes (e.g., the degree). Specifically, it is the following Pearson correlation coefficient [49]:

$$r = \frac{\sum_{xy} xy(e_{xy} - a_x b_y)}{\sigma_a \sigma_b} \tag{1.13}$$

where $a_x = \sum_y e_{xy}$ and $b_y = \sum_x e_{xy}$, e_{xy} is the fraction of links that connect nodes with value x to nodes with value y, σ_a and σ_b are the standard deviations of the a_x and b_y distributions respectively.

Another form of assortativity is the node-level Pearson's correlation coefficient between the degree and the *Average Nearest Neighbor Degree (ANND)* [50], or between the strength and the *Average Nearest Neighbor Strength* (ANNS) [51]. In particular, the ANNS is defined as:

$$\text{ANNS}_\text{i}^{\alpha/\beta} = \frac{\sum_j^{\mathcal{N}_i^\alpha} s_j^\beta}{k_i^\alpha} \tag{1.14}$$

where $\mathcal{N}_i^\alpha$ is the α neighborhood of i, and $\alpha, \beta \in [\text{in}, \text{out}]$. That is, it can be computed consider the different in-out link directions. The aggregated ANNS quantity can be defined if $\alpha = \beta = tot$ and the *in* and *out* neighbors are aggregated. For the ANND case, instead of the strength s_j^β one should consider the degree d_j^β of nodes.

Connected components A *connected component* is a sub-network such that there is at least a path between every pair of nodes. The largest of these components in a network is

often referred to as Largest Connected Component (LCC) or Giant Connected Component (GCC).

In the case of directed networks, a *strongly connected component* is defined as the sub-network such that there is at least a directed path between every pair of nodes. If directionality is ignored instead, we talk about *weakly connected components* to avoid confusion.

1.1.2.2 Centrality measures

One of the main topics of Network Science has always been trying to understand and quantify what makes a node important, often to *rank* the nodes in a network. However, considering that there is no unique definition of importance — and that the definition may be domain-dependent —, many *centrality measures* have been proposed, each with their own application.

In the following, we briefly introduce some of the most important centrality measures.

Degree centrality A simple centrality measure is the *degree centrality*. As the name suggests, it is degree-based and the importance of a node depends on its [in | out] degree. The rationale is that, for instance, in the network of the who-follows-who of online social networks like Twitter, the in-degree captures how many users follow another: the higher the in-degree, the more "important" that node is. Nodes with a large degree (w.r.t. the other nodes in the network) are often called *hubs*.

Betweenness centrality The *betweenness centrality* [52] is a measure based on shortest paths. Specifically, for each node i, it is the fraction of the shortest paths between any pair of nodes that pass through node i. Formally, a for node i it is defined as [53]

$$b_i = \sum_{s \neq t \neq i \in V} \frac{\sigma(s,t|i)}{\sigma(s,t)} \tag{1.15}$$

where $\sigma(s,t)$ is the number of the shortest paths from nodes s to t, and $\sigma(s,t|i)$ is the number of the shortest paths from s to t that pass thorough i.

The betweenness centrality can also be defined for links.

Eigenvector centrality The *eigenvector centrality* [54], also known as *eigencentrality*, is a spectral centrality measure that not only accounts for the number of connections but also for their quality, i.e., two nodes with the same number of connections will have different centrality values if their neighbors have different centrality values [55]. Formally, for a node

i it can be defined as [55, 56]

$$\lambda e_i = \sum_j^{\mathcal{N}_i^{\text{in}}} e_j = \sum_j A_{ji} \cdot e_j = (\mathbf{A}^\top \mathbf{e})_i \tag{1.16}$$

where $\mathcal{N}_i^{\text{in}}$ is the in-neighborhood of node i, and $\mathbf{A}$ is the adjacency matrix of the network. That is, the eigenvector centrality of node i is corresponding component of the eigenvector of the transpose of the adjacency matrix with the largest eigenvalue λ [55].

PageRank The *PageRank centrality* [57] was introduced in 1998 by Google's founders Page and Brin (and their coauthors) to assess the importance of web pages [58].

It is a centrality measure based on random-walks — i.e., on an agent that walks in the network by choosing, at every node, a random out-link to follow —, used to model the behavior of a random user surfing the Web. To avoid that the random-walker could get stuck in a sink node (i.e., a node without out-links), a *damping factor* probability q, usually set to a small value (typically 0.15), can make the random-walker jump (*teleport*) to a random node in the network. The teleport operation simulates the user moving to another Web page, e.g., by clicking on a bookmark. As in the eigenvector centrality, nodes with many "important" neighbors are more important than other nodes with the same in-degree but less important neighbors [55].

The PageRank of a node i can be described as follows [55]

$$p_i = \frac{q}{N} + (1-q) \sum_j^{\mathcal{N}_i^{\text{in}}} \frac{p_j}{k_j^{\text{out}}} \tag{1.17}$$

where q is the damping factor, $\mathcal{N}_i^{\text{in}}$ is the in-neighborhood of node i, N is the number of nodes in the network, k_j^{out} is the out-degree of node j. In practice, it is computed iteratively using the power method.

For more about the PageRank centrality, we refer the Reader to the dedicated Section 4.3.1.

1.1.2.3 Community analysis

Community analysis has been proposed in the social sciences [59, 26] and consists in finding *communities*, i.e., sub-sets of nodes such that connections between the nodes in the same community are denser than with nodes in the other communities [60]. Moreover,

one might be interested in finding *overlapping communities*, where a node might belong to multiple communities with different belonging factor.

While finding optimal communities is a hard task — it is a combinatorial problem and the number of communities may not be known a priori —, many algorithms have been proposed. For instance:

- Girvan and Newman [61] compute edge-betweenness iteratively to find and remove edges that lie between communities (the rationale being that edges with high betweenness bridge many nodes) and analyze the connected components after every removal until no edge remains. The output of the algorithm is a dendrogram (a hierarchical tree) with the complete community structure of the network;

- Blondel et al. [62] propose a *modularity* maximization algorithm, known as *Louvain* method. In particular, while modularity Q was already proposed by Newman [63] to evaluate the quality of partitions in weighted networks as

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{1.18}$$

  where $m = |E|$ is the number of links, k_x and c_x are the strength and the community of node x respectively, and $\delta = 1$ if $c_i = c_j$, Blondel et al. optimize this quantity with two phases repeated iteratively. Specifically, they begin by placing each node in its own community. Then, the first phase consists in considering the modularity gain obtained by moving each node in the neighboring communities. The node is placed in the one that provides the maximum positive gain, if any, or is left in its community otherwise. In the second phase, they aggregate the communities found in the previous step and generate a new network where each community is a node and edges are assigned a weight that corresponds to the sum of weights between the two communities. Self-loops account for internal links instead. In their work, Nicosia et al. [64] extend the definition of modularity to overlapping communities;

- Peixoto [65, 66] employs Bayesian inference, a method of statistical inference, to fit the parameters of generative models like the Stochastic Block Model proposed by Holland et al. [67] and its variants, and find the ones that most likely have generated the network (and its community structure).

1.1.3 Network generation

In many applications, the generation of synthetic/random data with some given properties can be useful, for instance to benchmark an algorithm or to create a *null model* — i.e., an ensemble of random graphs with given characteristics (like degree distribution, etc.) that is taken as a term of comparison for other structural properties, in order to verify if the results obtained on the analyzed data are meaningful or not —, like in community detection.

In the following, we briefly discuss some of the most popular generative models.

1.1.3.1 Erdös-Rényi (ER)

The Erdös-Rényi (ER) models, proposed by Erdös and Rényi [68], can be defined using two *static* models, meaning that the set of nodes does not change over time [26]. The first model, denoted with $\mathscr{G}^{\text{ER}}_{N,K}$ [26], generates a random network with N nodes and K (randomly) placed links by staring with a disconnected network and connecting K random node pairs. The second model, denoted with $\mathscr{G}^{\text{ER}}_{N,p}$ [26], instead of focusing on the number of links, generates a network where each pair of nodes is connected a given with probability p. In both cases, each link is present (or absent) with equal probability. Therefore, the degree distribution is binomial [26].

1.1.3.2 Barabási-Albert (BA)

Barabási and Albert [34] proposed a generative model, called *Barabási-Albert* (BA), that mimics the dynamic growth of many observed real-world networks, like the World Wide Web [69]. The model, denoted with $\mathscr{G}^{\text{BA}}_{N,m,m_0}$, is based on two mechanisms [26]:

- Growth: new nodes are added during the network generation until the desired number of nodes N is reached, i.e., the resulting networks are the result of a growth process;
- Preferential attachment: new nodes tend to link to nodes with higher degree. This process was inspired by the observations of new web-sites in the World Wide Web network [69] that link to the most popular ones.

In detail, the Barabási–Albert algorithm starts with a network with m_0 isolated nodes. New nodes are created one at a time, and each is connected to $m \leq m_0$ other nodes. The probability that the new node is connected to an existing node u is $p_u = \frac{k_u}{\sum_j k_j}$.

It has been proven that, for $N \to +\infty$, the output degree distribution follows a power-law $P(k) \propto k^{-\gamma}$ with $\gamma = 3$ [26].

The model can be extended to directed networks by generating the *inlink* and the *outlink* distribution separately.

Some variants of the BA model have been proposed, like the Bianconi-Barabási model, where the preferential attachment is also influenced by a static value called *fitness*. In this model, nodes with higher fitness value are more likely to attract links than other nodes [70].

1.1.4 Network robustness

Robustness is the ability of a network to withstand failures in its structure, which can be the result of random failures or targeted attacks. Considering the importance of many real-world networks (e.g., infrastructure, social, biological, brain), the analysis of robustness has been a topic of broad and current interest since the seminal work by Albert et al. [71]. In particular, there are two types of robustness [26] that can be studied as nodes (or edges) are removed: *static robustness*, i.e., how the topology and/or some properties of the network change, and *dynamic robustness*, i.e., how the flow of some quantity (e.g., information, power, etc.) changes. In the following, we briefly introduce the Reader to the topic and refer to Chapter 3 for more insights and references.

1.1.4.1 Static effects

The study of static robustness refers to the study of the change in network topology (or some of its properties) as some of its components are removed from the network. Specifically, we often refer to *site percolation* for the removal of nodes, and to *bond percolation* for the removal of links.

The seminal works in this direction is the one by Albert et al. [71] that study the robustness of synthetic Erdös-Rényi (ER) and Scale-Free (SF) networks, and also the robustness of the sub-network of the Internet and of the World-Wide Web. In their work, they employ the size of the Largest Connected Component (LCC) as a function of the removed nodes to evaluate the health of the system. The rationale is that, since most networks require the existence of a giant component to work properly, if the LCC is small enough the other components are even smaller, the system is expected to malfunction. Some examples are the Internet network, where being unable to reach another user in the network is considered a malfunction, and the power-grid system, where a component (e.g., a district) without power is not desirable. One of the most interesting results from their experiments is that, due to their uniform degree distribution, Erdös-Rényi networks are more robust to random failures (e.g., random hardware malfunction of servers) than networks with fat-tailed degree distributions, while real-world networks are more vulnerable to target attacks (e.g., to hubs).

1.1.4.2 Dynamical effects

If the nodes or the links of the network can tolerate at most a certain flow before failing, the removal of nodes or links may lead to consequent failures due to the redistribution of flows. That is, there might be dynamical effects that cannot be evaluated with a static analysis. This is the case, for instance, of power-transmission grid networks where each link can deliver at most a certain power and gets damaged in case of overload. Of course, each of these consequent failures may trigger more failures, and so on, and even a small initial shock may lead to an avalanche that is called *cascading failure* [72]. Popular examples of cascading failures are the North American and Italian blackouts that happened in 2003. For more about cascading failures and their modeling, we refer the Reader to other works [26, 73, 72, 74].

1.1.5 Network dynamics

In the previous sections we gave an overview on how the structure of complex systems can be analyzed using networks and the Network Science tools. However, another topic of great interest is the analysis of the network dynamics, i.e., how the structure and the state of the network (or of its nodes and links) changes over time. An example are cascading failures, already introduced in the previous section, but also other phenomena like spreading/diffusion and synchronization have been widely analyzed in the literature [25, 38, 73]. While out of the scope of this thesis, in the following we briefly introduce spreading phenomena to further motivate Network Science and the importance of networks.

1.1.5.1 Diffusion and spreading phenomena

One of the applications of Network Science is the study of the dynamics of spreading phenomena across different network topologies [25, 38, 73, 75]. In fact, being networks an agnostic representation of a system, links can model various mean of propagation (flow, contact, transfer, face-to-face interactions, etc.), and networks offer a common framework to study different phenomena. For instance, the spread of ideas and rumors can be studied on (online) social networks, the spread of malware can be analyzed on computer networks, contact networks can be used for the spread of pathogens, etc. [73].

Epidemic diffusion. Epidemic diffusion is one of the first and most widely analyzed diffusive phenomena [38] due to its importance for society, for instance, for the study immunization techniques (e.g., vaccinations) and how they affect the spread of diseases.

The simplest models used in epidemiology to forecast epidemics are the *Susceptible-Infected-Susceptible* (SIS) and the *Susceptible-Infected-Recovered* (SIR). These two models

are compartmental, i.e., each individual is assigned to a compartment that defines its status and that can change over time. For instance, in the SIS and SIR models, each individual can be classified as:

- *Susceptible (S)* if they have not contracted the pathogen yet;
- *Infectious (I)* if they are contagious;
- *Recovered (R)* if they recovered and have become immune to the disease.

These models have two parameters: the transmission rate, i.e., the probability that an infectious neighbor infects a node and that commonly denoted with λ, and the recovery rate, commonly denoted with μ, i.e., the probability that a node recovers from the disease (or dies). It is also common to define the ratio $\sigma = \lambda/\mu$ and to try to find the critical value σ_c, called *epidemic transition*, such that if $\sigma > \sigma_c$ the spreading may cause an epidemic [26, 73]. While the key difference between SIS and SIR models is that in the second, after recovering, individuals become immune to the disease, the associated dynamics are completely different [26]. For instance, in the SIR model any epidemic will eventually end if no new individuals join the network, because soon or later everybody will get immune, while in the SIS model individuals can get infected again.

For more details, we refer the Reader to [26, 73] and to the literature.

1.1.6 Multilayer Networks

While the classical definition of network allows two nodes to be connected by a link, in real-world systems entities can interact in multiple ways. For instance, in transportation networks, two nodes (areas) can be connected by multiple means (e.g., road, rail, tube, bus, etc.) [76, 77], two individuals may have different kinds of relationships (e.g., friendship, business, family, co-workers, business, sport, etc.) [78], or connect on multiple online social networks (Facebook, Twitter, etc.) where, in each, can interact in the various ways allowed by the platform (e.g., friendship, follow, share, like, mention, etc.).

The answer to this limitation are *multilayer networks*, where each type of connection is encoded in a different *layer*, i.e., a network with its own set of nodes and connections, called *intra-layer links*. For instance, the multilayer representation of the transportation network of a country could have a "roads" layer, where nodes are intersections and links represent the roads, an "air" layer, where nodes are airports and links model flights, a "bus" layer, where nodes are bus stations and links are bus rides, etc. However, in many scenarios the entities that belong to two different layers may interact, too. Such connections are called

inter-layer links. In the transportation network example, airports (nodes in the "air" layer) and bus stations may be connected by, e.g., taxis.

In the following, we briefly introduce the Reader to multilayer networks. For more thorough reviews about multilayer and interdependent systems and other examples of applications, we refer the Reader to [73, 79–89, 76] and references therein.

1.1.6.1 Definition and types

A *multilayer network* $\mathscr{M}$ is an ordered pair $\mathscr{M} = (\mathscr{G}, \mathscr{C})$ where $\mathscr{G} = \{\mathscr{G}_\alpha, \forall \alpha \in \{1, \ldots, L\}\}$ is a set of L networks, called *layers* of $\mathscr{M}$, and C is the set of interconnections, each called *crossed layers*, between the nodes of two different layers [73]. The links between two nodes in the same network are called *intra-layer* connections, while the links between nodes in different networks are called *inter-layer* connections. Of course, the meaning of each layer and of the inter- and intra- layer connection depend on the domain of the modeled system.

Multilayer networks are the generalization of various mathematical objects [73] that can be considered sub-types. For instance:

- *Multiplex networks* [90], also known as *edge-colored graphs*, share the same set of nodes V in all layers. These networks are useful to model systems that have different kinds of connections among nodes, but no inter-layer connectivity can be inferred from data, like in online social networks (where users can connect through Twitter, Facebook, etc.), or where links have different physical meaning (e.g., co-activation at different frequencies, etc.);

- *Temporal networks*, where each layer is the representation of the system at a different time t, with t being an integer. Inter-layer connections are allowed only between contiguous layers for temporal contiguity. In other words, only nodes in layers t and $t+1$ can be connected.

1.1.6.2 How to study multilayer networks

A simple way to study systems characterized by multiple relationships is to consider a single network (or monoplex) that is the result of the aggregation of the different kinds of relations. While this approach has often been used in the past, it has multiple serious issues and received a lot of criticism, since it is inherently affected by loss of potentially essential information about the structure of the system and, consequently, about its function. For example, some open questions are: *how to aggregate the layers?*, and, *is it correct to do so?*. The answer to the first question requires the definition of an aggregation function

(sum, average, min–max, etc.) and a proper scaling of the weights associated to each kind of relation. The second question poses an even more important problem: does it conceptually make sense to aggregate? The answer, in general, is negative, since often relations encode different contexts that cannot be simply mixed together without altering the structure or the function of the system under investigation [90, 91, 76, 92, 93, 82, 94, 95].

1.1.6.3 A new mathematical framework

De Domenico et al. [80] developed a new *ad hoc* mathematical framework to study such networks. In particular, they represent a multilayer network as a higher-order adjacency matrix, specifically a rank-4 tensor, where each entry $M_{j\beta}^{i\alpha}$ represents a link from node i in layer α to node j in layer β. That is, if $\alpha = \beta$ the link is intra-layer α, and inter-layer otherwise. Thanks to this framework and by exploiting tensorial algebra, many classical network descriptors can be generalized, from centrality measures to community detection. It should be noted that multiplex networks require a rank-3 tensor for their mathematical representation [96, 92], since $M_{j\beta}^{i\alpha} = 0 \; \forall \alpha \neq \beta$.

In practical applications, it is common to flatten the $\mathscr{M}$ into lower-rank tensors, named *supra-matrices* [88, 80], via matricization [97]. This operation consists in mapping the tensor $\mathscr{M} \in \boldsymbol{R}^{N\times L\times N\times L}$ to a rank-2 tensor in $\mathbf{A}_{\mathscr{M}} \in \boldsymbol{R}^{NL\times NL}$ where diagonal blocks encode single-layer connectivity, while off-diagonal blocks encode cross-layer relationships.

1.1.6.4 Application scenarios

Just like classic networks ones, multi-layer ones find application in many scenarios. For instance:

- In neuroscience: De Domenico et al. [98] employ multiplex networks to identify schizophrenic patients. In particular, for each patient they build a multiplex brain functional network — i.e., a multiplex network where nodes are special brain regions that play a fundamental role in the brain functional connectivity, the links represent the connectivity strength estimated by inter-regional correlations calculated on processed fMRI (functional magnetic resonance imaging) signals, and each layer corresponds to a different frequency band [98] — and then use the multi-layer PageRank centrality to distinguish between healthy and schizophrenic patients;

- To study international trade systems, as they can be characterized by the presence of several kinds of relationships among countries, depending on the commodities they trade [99, 100];

- In computational biology, as multiple types of relationship among their constituents, such, for instance, in the Homo Sapiens proteome, where protein–protein relations can be of two types, physical and genetic, such as interactions, chemical associations or post-translational modifications [101–103];
- To model multiple types of transportation systems, e.g., each commercial airline can be mapped to a different layer [104];
- In epidemiology [105], e.g., to model multiple spreading pathogens or to study the dynamics of coupled processes.

1.2 Machine Learning

In this Section, we introduce some Machine Learning fundamentals that can help the Reader understand the motivations and techniques of Geometric Deep Learning, and also the applications proposed in this thesis. In particular, we introduce some terminology and applications, the most popular models, and Deep Learning along with its training methodologies and algorithms.

1.2.1 What is Machine Learning?

Machine Learning (ML) is the study of algorithms that extract information (*learn*) from data to improve their performance [106]. While the idea of a learning machine can be attributed to Touring [107] and the first neural networks and learning algorithms date back to the 1950s [108], the explosion in popularity of Machine Learning over the last decade is due to the rise of Deep Learning, a family of algorithms that outperform classical ones in many tasks, and to the large availability of data to learn from along to the computational power provided by modern hardware, allowing its processing. Another strong point of Machine Learning is that it is suitable for those tasks that are intuitive but too hard to be described in a formal way, like identifying and classifying objects in images, etc.

1.2.2 Applications

Machine Learning has been successfully employed in many applications, for instance:

- To build recommender systems, i.e., systems that predict the preferences of users. An example is the one built by Netflix [109];
- In Natural Language Processing (NLP), i.e., the branch that processes text or voice inputs trying to understand and answer. Examples are translators and virtual assistants;
- In computational biology, e.g., to predict protein fold structure with the AlphaFold [110] and AlphaFold2 [111] algorithms from DeepMind;
- To learn to play games, like chess and go (AlphaZero [112]), but also classic arcade games [113];
- To upscale images in games in real-time with NVIDIA DLSS (Deep Learning Super Sampling) [114];
- In many domains, to predict future events/values by learning trends on past ones;

- To approach various NP-hard combinatorial problems, like the maximum clique enumeration [115];
- To classify images, audio, video, text, etc.;
- For decision-making under uncertainty [116];
- For pattern recognition;
- To generate images, videos, audio, text, etc.;
- For anomaly detection.

1.2.3 Learning machines?

A formal definition of "learning" was provided by Mitchell [106], according to whom a computer program learns if its performance improves through *experience* with respect to some class of *tasks* and *performance measure*. Specifically, the *training* of an algorithm is meant to optimize its parameters (also called *weights*), and the output algorithm (along with its parameters) is called a *model* [117]. That is, each algorithm is a family of functions, and different models can be defined by different sets of parameters. It should be noted that algorithm's learnable parameters should not be confused with the *hyper-parameters*, the configuration provided by the user and that cannot be learned from data.

The data used to train the models is organized as a set of *examples* (or *observations*) with some attributes, called *features*. The set of the examples is called *dataset*, and it is usually represented in matrix form, where each row is an example and each column a feature.

It is worth mentioning that the main difference between Machine Learning and optimization is that the first aims at generalization performance, not only minimizing the loss function on the training data [117]. If the model does not *generalize* well to new observations, i.e., its performance is poor on previously unseen data, we say it is *over-fit*. If the model performs poorly even on the train data, we say it is *under-fit*.

1.2.3.1 Training methodologies and tasks

There are three main ways an algorithm can obtain experience from the dataset during the learning process [117]:

Supervised learning In *supervised learning*, each observation in the dataset comes with the correct *label* (i.e., the value or class). That is, our dataset has the form of a set of features

with labels: $\{(\mathbf{x}_i, \mathbf{y}_i)\}$. The objective of the training is to teach (supervised) the model to predict the correct label.

The most common tasks associated to supervised learning are:

- *Classification*, given an input and a fixed number n of classes, the model has to predict which class or classes the input belongs to. Binary classification is classification refers to classify between two classes;
- *Regression*: given an input, the model has to predict a real value;

Unsupervised learning In *unsupervised learning*, there are no labels associated to the examples, and the model has to, e.g., find and exploit patterns in the data. Of course, the performance evaluation of the model less trivial than the previous case. This kind of learning is suitable for various tasks, for instance:

- *Clustering*: given a set of inputs, clustering is about dividing observations into groups that share some property;
- *Pattern discovery*: find common patterns in the data;
- *Anomaly detection*: detect anomalies in the data (e.g., in intrusion detection systems, etc.).

Semi-supervised learning *Semi-supervised learning* is a hybrid methodology that falls between supervised and unsupervised learning. In fact, just a part of the dataset is labeled, and the model is trained both in a supervised and in an unsupervised way.

Reinforcement learning In *reinforcement learning*, there is no fixed dataset. Instead, the model is used to drive the behavior of a software *agent* that interacts with some (simulated) environment. The model is then trained by feedback: after an action (or a sequence of), it will be given a positive or negative *reward*.

1.2.3.2 Loss functions

According to the definition by Mitchell, the Machine Learning algorithms improve their performance with respect to some performance measure, that is optimized during the training phase [106]. This performance measure is called *loss function* (or *objective/error function* [117]). The loss functions are task-dependent, but, in general, the aim of the training

is to reduce their value, as this reduction translates in a reduction of the error of the model (commonly called *loss*).

Some of the most common loss functions include:

- The *Mean Squared Error* (MSE) function, used in regression tasks. It is formally defined as:

$$L_{\text{MSE}} = ||y_i - x_i||^2 \tag{1.19}$$

  It should be noted that the square of the error reduces smaller errors while assigns larger weights to larger ones;

- The *Cross Entropy*, used in classification tasks, formally defined as [118]:

$$L_{\text{CE}} = -\frac{1}{N}\sum_{i}^{N}\sum_{c=1}^{M} y_{i,c}\ \log(p_{i,c}) \tag{1.20}$$

  where M is the number of classes, $y_{i,c}$ is the binary indicator $(0,1)$ if the predicted class is correct, and $p_{i,c}$ is the predicted probability that observation i belongs to class c.

  In binary classification tasks this becomes:

$$L_{\text{BCE}} = -(y\ log(p)) + (1-y)\ log(1-p) \tag{1.21}$$

  also called *Binary Cross Entropy*.

It is also common to add to the loss a *regularization* term R that accounts for the magnitude of the model's parameters to avoid over-fitting [117].

$$\text{Loss} = \text{Loss}_{\text{task}} + \lambda R(\mathbf{W}) \tag{1.22}$$

where λ is the regularization parameter (or *weight decay*) and $\mathbf{W}$ are the model parameters.

The most common R functions areg [117]:

- The L^1 norm: $R_{absolute}(\mathbf{W}) = ||W||_1 = \sum_i |w_i|$;
- The L^2 norm: $R_{square}(\mathbf{W}) = ||W||_2 = \sqrt{\sum_i |w_i|^2}$.

1.2.3.3 Parameters optimization

As previously discussed, the optimization of the parameters of a Machine Learning algorithm is meant to improve its performance. However, a thorough exploration of all the

parameter combinations is not feasible even for simple algorithms, as the parameters are real values. One way to tackle this problem relies on specific algorithms that change the parameters iteratively. Unfortunately, the main drawback is that the loss function may not reach the global minimum but some local one.

The most common family of optimization algorithms is the one of the *Gradient Descent* [117]. The basic idea of Gradient Descent algorithms is to change the parameters iteratively moving in the direction of the negative gradient of the loss, since the gradient is the direction of the maximum increase in the function, one can use the negative gradient to approach some minimum. Specifically, the parameters are first initialized randomly (many initialization techniques exist, but we do not cover them here for simplicity), and then the algorithm's parameters are updated to follow the slope of the negative gradient of the loss, aiming for the global minimum. An iteration over the entire dataset is called *epoch*.

Stochastic Gradient Descent (SGD) The *Stochastic Gradient Descent* (SGD) is the simplest version of the algorithm, and generates the updated parameters $\tilde{\mathbf{W}}$ after each training example as follows [117]:

$$\tilde{\mathbf{W}} = \mathbf{W} - \varepsilon \Delta \mathbf{W} \tag{1.23}$$

$$\Delta \mathbf{W} = \nabla_{\mathbf{W}} L(f(x_i, \mathbf{W}), y_i) \tag{1.24}$$

where $\mathbf{W}$ are the model parameters, ε is the *learning rate*, L is the loss function, f is the model, x_i is the training example and y_i its label (assuming that a supervised task is being optimized). Note that the learning rate plays a key role, and many more advanced optimization algorithms adjust it with the epochs to improve performance.

While Stochastic Gradient Descent updates the algorithm's parameters after each observation, which may be inefficient and cause instabilities in the parameters, multiple variants have been proposed to improve performance and to mitigate its issues.

Batch Gradient Descent In *Batch Gradient Descent*, the gradient of the loss is computed for all the training examples and is averaged before updating the parameters. This reduces instabilities but also slows the learning process. Formally, the $\Delta \mathbf{W}$ is defined as:

$$\Delta \mathbf{W} = \frac{1}{N} \cdot \nabla_{\mathbf{W}} \sum_{i}^{N} L(f(x_i, \mathbf{W}), y_i) \tag{1.25}$$

where N is the number of examples in the training set (i.e., only one update per epoch).

Mini-batch Gradient Descent *Mini-batch Gradient Descent* is the middle ground between Stochastic Gradient Descent and Batch Gradient Descent as the training examples are divided in *mini-batches* (i.e., sub-sets), and the algorithm's parameters are updated with the average of the loss of each one. The number of updates per epoch is N mod B, where B is the number of examples in each mini-batch. Formally, $\Delta\mathbf{W}$ is defined as:

$$\Delta\mathbf{W} = \frac{1}{B} \cdot \nabla_{\mathbf{W}} \sum_{i}^{B} L(f(x_i, \mathbf{W}), y_i) \tag{1.26}$$

Momentum-based variants Other variants consider, for instance, the *momentum* of the parameters updates to converge faster. Specifically, the $\Delta\mathbf{W}$ is computed as the weighted sum of the previous step and the new gradient.

1.2.3.4 Back-propagation algorithm

An extremely important ingredient in the training of Machine Learning algorithms is the *back-propagation* algorithm, as it allows to efficiently compute the gradient of the loss in a feed-forward network [117]. However, being this topic out of the scope of this thesis, we refer the Reader to [117] for the details about its rationale and about how it works.

1.2.3.5 Performance evaluation

The loss function used during the training phase can be hardly understood by humans when evaluating the model, and does not give any information about the generalization performance. There are many possible choices for the performance measure, and it should be stressed that the right one is task specific as some outcomes may be more desirable than others [119] in certain domains, as not all errors made by the model have the same consequences. This is the case, for instance, of medical applications, where, e.g., a false negative COVID-19 test might lead to many infected people, while a false positive test just leads to a second test and quarantine for a single person.

Here, we report some of the most common performance measures. Note that, while these definitions are for single-label or single-class tasks, it is easy to generalize them to multi-class ones. For example, the final average score can be computed as the average of the score of each label.

Classification tasks *Accuracy score* is used for measuring the raw number of correct predictions. This score is suitable for tasks where all errors have the same severity. For

binary classification tasks, accuracy is defined as:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{TN} + \text{FP}} \tag{1.27}$$

Precision is a measure of how many correct positive predictions have been made.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1.28}$$

Recall (also known as *sensitivity*, *hit-rate* or *True Positive Rate* (TPR)) is a measure of how many correct positive predictions have been made, over all the positive cases.

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{1.29}$$

Specificity is a measure of how many correct negative predictions have been made, over all the negative cases.

$$\text{specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{1.30}$$

Fall-out (or *False Positive Rate* (FPR)) is a measure of how many wrong positive predictions have been made, over all the positive cases.

$$\text{fallout} = \frac{\text{FP}}{\text{FN} + \text{TP}} \tag{1.31}$$

F1-score is a measure that combines both precision and recall in their harmonic mean. Both scores account equally and must be high for F1 to be high.

$$F1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{1.32}$$

Area Under the Receiver Operating Characteristic Curve (AUC ROC) is the area under the sensitivity vs. fall-out curve at various discriminatory threshold levels. The AUC is a value between 0 and 1. Specifically, if AUC $= 1$, the classifier is perfectly distinguishing between the true positive and true negatives; if AUC $= 0.5$, the classifier is making random predictions. Example ROC curves are shown in Figure 1.2.

Regression tasks If small errors account just as large ones, *Mean Absolute Error* (MAE), defined as:

$$\text{MAE} = \sum_i |y_i - p_i| \tag{1.33}$$

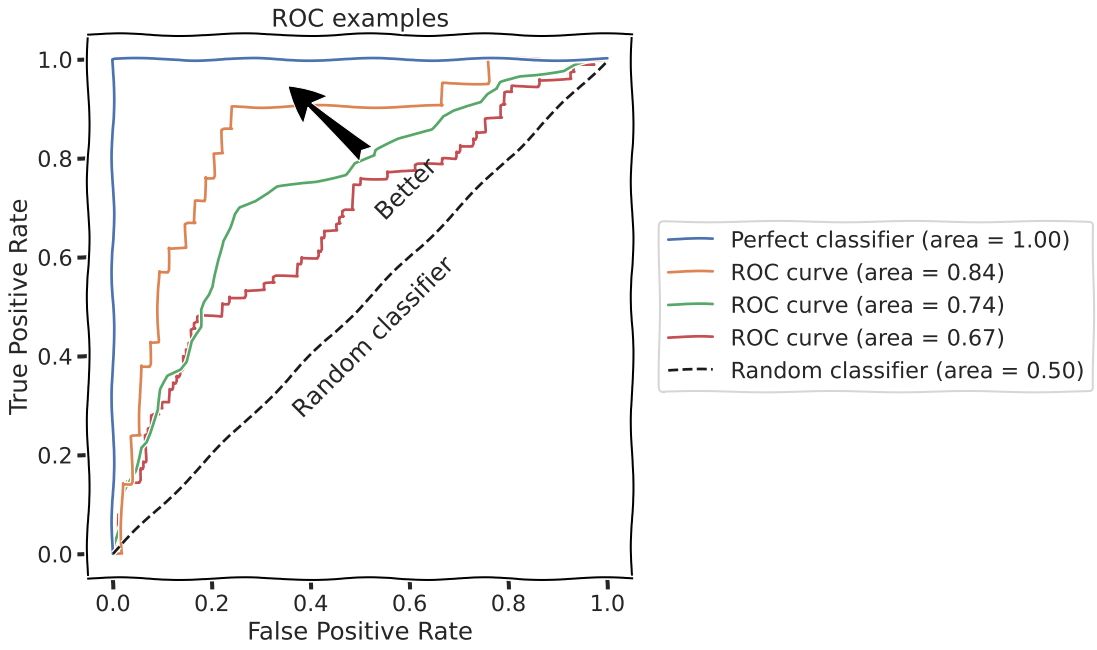


Fig. 1.2 **Example of ROC curves.** Each curve is obtained by changing the discrimination threshold on the same predictions. The larger the area, the better.

If large errors should account more, *Mean Square Error* (MSE)

$$\text{MSE} = \sum_i (y_i - p_i)^2 \tag{1.34}$$

1.2.3.6 Dataset splits

In Machine Learning the focus is on generalization performance, i.e., in evaluating the trained model on previously unseen data to verify if it is over-fit or if it has really learned the task. This requires at least two sub-sets of the full dataset:

- A *training set* with the observation used to train the model (often about $70-90\%$ of the full dataset);
- A *test set*, with the observation used to validate the generalization performance (often about $30-10\%$ of the full dataset).

However, to try to actively counter the over-fit phenomenon, it is also common practice to create a third sub-set, called *validation set*, used during the training phase as "test-set" to stop the training and choose the best model parameters without biasing the test phase. In other words, while no training is performed on this set, it is used to select the parameters that generalize better and avoid over-fit.

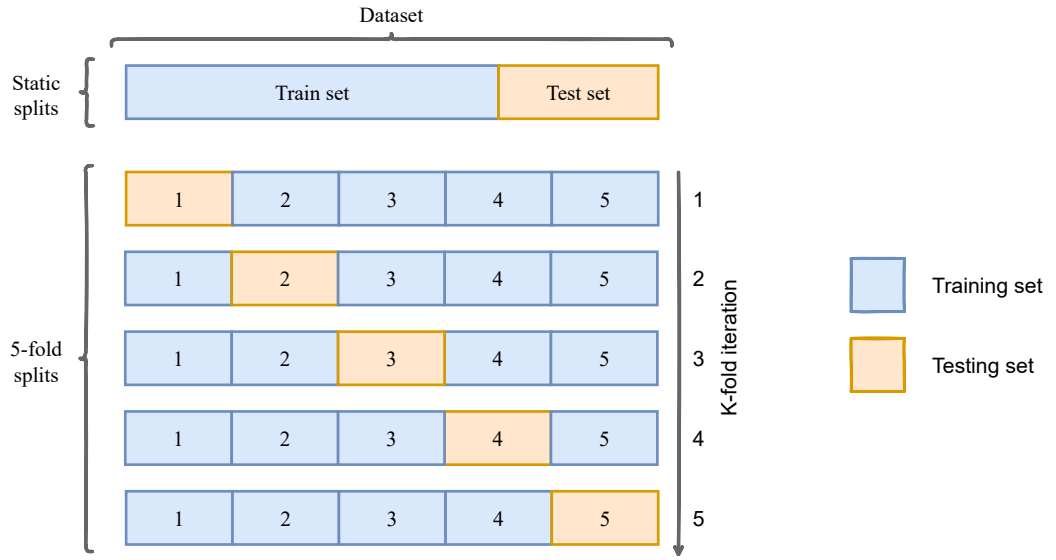


Fig. 1.3 **K-fold vs static split.** Representation of the K-fold splits (with $k = 5$) of a dataset and comparison with a static split. In the k-fold setting, k models are trained and the results on the test set are averaged.

Cross-validation While the static training/test set splits work for many tasks, in the case of smaller datasets one might be interested in maximizing the data efficiency. One technique for this purpose is *Cross-Validation* (CV). In its basic form, called k-fold Cross-Validation, the dataset is split in k sub-sets (called *folds*): the algorithm is trained on $k-1$ sub-sets and the remaining one is used as test-set; the procedure is repeated k times picking a different test sub-set every time and the results are averaged. Of course, this is computationally expensive but should give more informative scores. Taking k-fold CV to the extreme, we can also define the *Leave One Out* (LOO) cross-validation, where the number of splits is equal to the number of observations, i.e., $k = N$. *Stratified* cross validation makes sure that all the folds have the same percentage of samples of each class.

We show a visual comparison of K-fold and static splitting in Figure 1.3.

1.2.4 Models

Many Machine Learning algorithms have been proposed and used to solve various tasks. Examples are linear regression, decision trees, Support Vector Machines (SVMs), random forests and Artificial Neural Networks, and so on. Here, we briefly introduce the Reader to Artificial Neural Networks as they are now the most commonly used in many tasks like Computer Vision ones, and are also employed in the applications proposed in this thesis.

1.2.4.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are Machine Learning models inspired by the biological neural networks.

Specifically, biological neurons receive an input from other neurons via the dendrites and, if the input is strong enough, activate creating an output signal that is propagated via the axon. Artificial neurons are an approximation of such biological neurons that were first used to study the learning process in the brain [117]: they perform a weighted sum of the input, add a bias and apply a non-linearity, called *activation function*, that simulates the activation of the biological neuron. Formally, an artificial neuron performs the following operation:

$$y(\mathbf{x}) = \varphi(\mathbf{w}^T\mathbf{x} + \mathbf{b}) \tag{1.35}$$

where $\mathbf{x}$ and y are the input and output respectively, $\mathbf{w}$ are the parameters and $\mathbf{b}$ is the bias, φ is the activation function. That is, each neuron represents a linear sum plus an activation function, and has the same equation of a Support Vector Machine. Different artificial neurons can be defined using different activation functions. For instance, the first artificial neuron proposed is the *Perceptron*, where φ is the sign function (or some other binary function that produces an output according to a threshold), and the *sigmoid neuron* where the sigmoid function σ is used [120].

Artificial Neural Networks, called Neural Networks for simplicity, are networks of artificial neurons organized in *layers*. If the connections between neurons are acyclic, the network is said to be *feed-forward*, *recurrent* otherwise. A common Neural Network architecture is the *Multi-Layer Perceptron*, where all the neurons in a layer are connected to all the neurons in the next one. In such neural networks, the first and last layers are called *input* and *output* layers respectively, while the ones in the middle are called *hidden* layers.

1.2.4.2 Activation functions

In general any differentiable non-linear function can be used as an activation function, but some have shown better performance in different tasks. Each activation function comes with their pros and cons that, e.g., come from their derivative (computed during the back-propagation steps). Some examples are:

- The sigmoid (σ), that yields a value between 0 and 1, which is useful, e.g., to predict probabilities.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{1.36}$$

- The tanh, that returns a value between $-\frac{1}{2}$ and $\frac{1}{2}$, useful, e.g., to predict zero-centered values.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.37}$$

- The softmax, is a normalized exponential function that, given an input with N real numbers, returns an output with the same size where each value is proportional to the normalized exponential of the corresponding input. That is, it returns a probability distribution that can be used, for instance, when one needs to predict the probability of each of N classes.

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j^N e^{x_j}} \tag{1.38}$$

- The *Exponential Linear Unit* (ELU):

$$\text{ELU}(x) = \begin{cases} \alpha \cdot (e^z - 1) & x \leq 0, \\ \alpha x & \text{otherwise} \end{cases} \tag{1.39}$$

- The *Rectified Linear Unit* (ReLU):

$$ReLU(x) = max(x, 0) \tag{1.40}$$

- The *Leaky Rectified Linear Unit* (LeakyReLU), a variant of the ReLU, parametrized by α.

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0, \\ \alpha x & \text{otherwise} \end{cases} \tag{1.41}$$

- The sign function:

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x > 0, \\ -1 & \text{otherwise} \end{cases} \tag{1.42}$$

1.2.5 From Machine to Deep Learning

Deep Learning is a class of Machine Learning algorithms that have many layers, hence the name *deep* neural networks, to learn representations. Specifically, with *representation learning* we refer to learning a representation of the input data [121, 117] that is suitable for the task. This is a completely different paradigm from classical Machine Learning,

where hand-crafted features are designed with effort by, e.g., feature engineers. The strong advantage of deep neural networks is that they learn hierarchical representations [117]: each layer is able to combine the (output) features of the previous layer to build increasingly complex, higher-level features [122]. For instance, Lee et al. [122] show that, in a three layers deep neural network, the first layer learns to spot low-level features in images (e.g., oriented edges), the second layer uses such features as bases and combines them to form object parts, while the third layer leans task-specific objects (e.g., cars, faces, etc.).

1.2.5.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are neural networks with convolutional layers that are used to process data with grid-like topology, like images [117]. Convolutional Neural Networks were designed in the context of computer vision with the intuition of employing the *convolution* operator to process the images, and have become extremely popular thanks to their performances in many scenarios.

The convolution operator between an input array $\mathbf{I}$ and a learnable array $\mathbf{K}$, called *kernel* or *convolutional filter*, produces an output array where each position x, y (assuming the input data and the kernel have two dimensions) is the sum of the dot product of *local receptive field* — i.e., that portion of the input that overlaps with the kernel when it is centered at x, y — with the kernel's parameters. This operator has many desirable properties, for instance [117]: 1. It allows parameter sharing, as the same kernel parameters are used for every input position; 2. It has sparse interactions since each position is computed via a few operations as the kernel is usually small with respect to the image. That is, the convolution is efficient and scales well to large images; 3. It is equivariant to translation (but not to some other transformations); 4. It processes image patches, thus it can handle inputs of variable size.

Pooling When performing multiple convolution steps in a deep model with many convolutional layers, it is common to shrink the output of the hidden layers to improve the computational time. It was shown that this operation does not degrade performance, thanks to the hierarchical representation learning. In fact, in the case of images, while the inputs are just arrays of pixels, as we get deeper in the network the values become high-level features. In particular, *pooling layers* produce new images by dividing the input image in (disjoint) image patches and by applying a (deterministic) function that returns a single pixel for each patch [117]. Common pooling functions are the max and the mean.

1.2.5.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are particular feed-forward neural networks with feedback connections [117], used to process sequential data, i.e., when the output should be affected not only by the current input but also by the previous ones and by their order. This is achieved introducing some internal state to capture information about the sequence that is being fed to the network.

Usually, models only have one recurrent layer. However, considering that its internal state is updated after each input, the actual number of layers is larger and equal to the sequence length, because of the cyclic connections that can be unfolded by applying the definition iteratively. It is important to note that the unfolded layers share the same parameters [117], which affects the training phase.

Recurrent Neural Networks are affected by various issues that make their training hard. For instance, the vanishing gradient, i.e. the gradient associated to some input becomes smaller and smaller after each iteration. In other words, after some time-steps, the network looses some context information about previous inputs, that will not be able to affect the output. Various RNN layers have been proposed to improve performance and address these issues [120], like e.g., Long Short-Term Memory (LSTM) [123] and Gated Recurrent Unit (GRU) [124], and most automatically learn whether to remember or forget some information in the internal state.

Chapter 2

Geometric Deep Learning

In this Chapter we introduce Geometric Deep Learning and its state-of-the-art. In particular, we focus on learning on graphs, the main topic of this thesis, and after a brief overview of historical (and outdated) approaches, we detail the intuition behind the Graph Neural Networks (GNNs) and their message passing architecture, overview some state-of-the-art GNN layers, and show some example applications. For more about Geometric Deep Learning, we refer the Reader to [125, 126] and the references therein.

Please note that, to avoid confusion between neural networks and networks (graphs), in this paper we will refer to the first as "neural networks" and to the latter as networks/graphs, and to avoid confusion between model layers and network layers, we will refer to each by specifying the context.

2.1 What is *Geometric Deep Learning*?

Geometric Deep Learning is a recent sub-field of Machine Learning that attempts to generalize Deep Learning models to non-Euclidean domains like graphs and manifolds [127, 128]. In fact, while classic Deep Learning algorithms are suitable for Euclidean data — i.e., data that can be mapped to Euclidean spaces (or to grids), like images, audio and vectors in general —, there is no way to employ such algorithms for data without an underlying Euclidean structure due to the lack of many properties (like a common system of coordinates and shift-invariance), and any non-natural mapping to a Euclidean space would lead to loss of information. However, as discussed in the previous Chapter, many interesting research fields and applications involve non-Euclidean data like graphs, and the lack of suitable algorithms led Machine Learning researchers to define new ways to process and learn on such data.

It is also worth mentioning that Hamilton tracks Geometric Deep Learning's roots to two more concepts, developed independently [125]: to a differentiable variant of belief

propagation [129], and to the Weisfeiler-Lehman (WL) algorithms for graph isomorphism test [130].

2.2 Applications

Geometric Deep Learning algorithms offer new ways to approach many problems on graphs, including Network Science and computationally hard ones. In particular, while most of the tasks that can be performed are the same as in the Euclidean case (e.g., classification, regression, clustering, etc.), different applications require different levels of granularity (i.e., node, link or graph-level). That is, one may need to classify a node, a graph or a link, cluster them, and so on. These tasks find application in a wide range of domains, as proven by the huge number of publications that employ Geometric Deep Learning to solve theoretical and real-world problems. Just to mention a few, Geometric Deep Learning has been successfully used:

- For drug discovery and development by Gaudelet et al. [131];
- To predict poly-pharmacy side effects by Zitnik et al. [132];
- To build recommender systems, like *PinSage*, deployed at Pinterest, by Ying et al. [133];
- To improve the robustness of networks [134];
- To detect fake news on social media by Monti et al. [135];
- For chip placement by Mirhoseini et al. in [136] and [137].
- For community detection, by Chen et al. [138];
- For quantum physics, e.g., for particle track reconstruction [139], and for quantum chemistry [140];
- To generate new graphs with learned characteristics [141–143];
- To predict traffic for Google Maps, by DeepMind [144];
- To approach various NP-hard problems [145, 146];
- For link prediction, even on temporal graphs [147]

- To simulate a various challenging physical domains, involving fluids, rigid solids, and deformable materials interacting with one another [148], and to learn mesh-based simulations and improve the efficiency of complex modeling tasks [149], like the prediction of the dynamics of a wide range of physical systems, including aerodynamics, structural mechanics, and cloth.

2.3 Graph Representation Learning

Geometric Deep Learning is often referred to as *"Graph Representation Learning"* [125]. The rationale behind this name is that, like their Euclidean counterpart, these algorithms aim to automatically learn from the data what better represents nodes, links or graphs for a given task, instead of requiring hand-engineered features. In fact, even with expert knowledge to design and pick the most relevant ones, the feature engineering process is hard and time-consuming, and may even not capture the best characteristics for the specific task. For instance, given a social network, what are the best features that characterize a user or one of its relations? The answer may not be trivial, especially in a complex network with billions of nodes and connections. Furthermore, the best features are often task and/or domain specific and may not generalize well. These limitations translate into long design time and high costs, and often poor performance.

2.4 Learning graph representations

The preferred way of most works in the literature to learn representations on graphs is to learn *node embeddings* using an *encoder-decoder* architecture.

In Deep Learning, an *embedding* is a learned and continuous low-dimensional vector that represents some entity [150, 117]. Specifically, it is a projection into a d-dimensional *latent space*, where the embedding of entities (the latent variables) should be closer together, if the entities are similar [151, 125, 117] with respect to a given task. While embeddings may not be directly interpretable, when learned correctly, they are sufficient to describe the original data, i.e., they are meaningful representation of that data. Embeddings are commonly used in Deep Learning, for instance, when dealing with words in Natural Language Processing [152, 153], with discrete variables, with high-dimensional vectors, etc.

In the context of graphs, the objective of Geometric Deep Learning is to map each node to a vector with d-dimensions — its embedding into a Euclidean space — that encodes some desirable properties [125]. It should be noted that, being the embedding a point in a

Euclidean space, it can now be fed to classic Machine and Deep Learning models to perform many tasks like regression or classification.

Most of the proposed approaches in the literature are a form of an *encoder-decoder* architecture, whose components are [125]:

1. An *encoder* $\text{ENC}(u)$ that maps nodes to embeddings. The encoder's parameters are trained to preserve some similarity in the embedding space;
2. A *decoder* $\text{DEC}(u)$ that reconstructs some task-related information about the nodes from their embedding. In many works, a *pairwise* decoder $\text{DEC}_{\text{pairwise}}(ENC(u), ENC(v))$ is used, which takes as input a pair of node embeddings to reconstruct some relation between the two;
3. A *loss function* to quantify the error on the decoded embeddings and train the model parameters.

The main difference between the various approaches is in the way they define those components.

2.4.1 A bit of history: *shallow* embeddings

Shallow embeddings are one of the most rudimentary attempts at learning on graphs. While they are important for historical reasons, these approaches have intrinsic limitations and are now considered obsolete.

Idea. The idea of *shallow* embeddings is to learn a matrix $\mathbf{Z} \in \mathbb{R}^{|V| \times d}$, where each row $\mathbf{Z}[u]$ is the embedding of node u and $|V|$ is the number of nodes in the network, to minimize the reconstruction loss over some similarity function [125].

Specifically, $\mathbf{Z}$ is optimized using an *encoder-decoder* architecture [125], where:

- The *encoder* $\text{ENC}(u)$ is just a lookup based on the node ID, i.e. $\text{ENC}(u) = \mathbf{Z}[u]$;
- The *decoder* $\text{DEC}(u)$ is a deterministic function that decompresses some information about the node from the learned embeddings.

As an example, in the case of pair-wise decoders, the shallow-embedding architecture can be optimized to minimize the reconstruction loss function so that:

$$\text{DEC}_{\text{pairwise}}(ENC(u), ENC(v)) \approx \mathbf{S}[u, v] \tag{2.1}$$

where $\mathbf{S}[u,v]$ is some similarity function between the nodes u and v. For instance, a pairwise decoder can learn to reconstruct the topology of the graph by predicting the existence of links between each pair of nodes.

$$\text{DEC}_{\text{link}}\colon \mathbb{R}^d \times \mathbb{R}^d \to \{0,1\} \tag{2.2}$$

$$\text{DEC}_{\text{link}}(\text{ENC}(u),\text{ENC}(v)) = \mathbf{A}[u,v] \tag{2.3}$$

Laplacian eigenmaps *Laplacian eigenmaps* [154] have been proposed by Belkin et al. in 2001. In particular, they define a pairwise decoder as the L^2 distance between the node embeddings $\text{DEC}(u,v) = ||\mathbf{z}_u - \mathbf{z}_v||_2^2$ and a loss function (for each pair of nodes u,v) $L_{u,v} = \text{DEC}(u,v) \cdot \mathbf{S}[u,v]$ that penalizes similar nodes with distant embeddings [125].

Inner-product methods A common class of shallow embedding methods defines the decoder as the inner-product between the learned embeddings [125]. That is, the decoder has form $\text{DEC}(u,v) = \mathbf{z}_u^\top \mathbf{z}_v$: the larger the dot product between the two embeddings, the more similar the two nodes are. Various methods differ in the similarity measure learned, and most use the Mean Square Error (MSE) as loss function. For example, *GraRep* [155] defines the similarity using powers of the adjacency matrix, while *High-Order Proximity preserved Embedding (HOPE)* [156] relies on general similarity measures (e.g. dice).

Random-walk based methods A more efficient class of shallow embedding methods relies on random walks to learn the embeddings [125]. For instance, *node2vec* [157], while still using the inner product decoder, combines Breadth-first Sampling (BFS) with Depth-first Sampling (DFS) in a flexible way in the encoder, using biased random walks to capture both local and higher-order properties of the network nodes.

Multilayer extension A possible generalization of shallow embeddings to multilayer networks is *OhmNet* [158]. OhmNet applies the node2vec algorithm discussed above to encode the neighborhood of nodes, and then encodes any dependency between the layers in a hierarchical way, where the hierarchy is represented using a directed tree $\mathscr{M}$.

Specifically, OhmNet tries to solve the following maximum likelihood problem:

$$\max_{f_1,\ldots,f_M} \sum_{l\in L} \Omega_l \;\; - \lambda \sum_{j\in\mathscr{M}} C_j \tag{2.4}$$

where L is the set of layers, and the two components are:

- Per-layer task-related objectives Ω_l, independent of each other, which estimate the node embedding in each layer l and make similar nodes in the same layer close together in the embedding space. This is achieved using the *node2vec* algorithm;
- Hierarchical dependency objectives C_j, which make nodes in nearby layers close together in the embedding space. This is achieved using a regularization term computed for each node $u \in \mathscr{M}_i$ where $\mathscr{M}_i$ is the set of the layers that belong to the sub-hierarchy rooted at i. Specifically, the regularization term, which is used to make sure the representation of the node u in each element of the hierarchy is similar to the parent's, has form:

$$c_i(u) = \frac{1}{2}|| f_i(u) - f_{\pi(i)}(u)||_2^2 \tag{2.5}$$

  where π is the parent-child mapping of the hierarchy, such that $\pi(i)$ is the parent of i. C_j is just the sum of c_i for every level of $\mathcal{M}$:

$$C_j(u) = \sum_{u \in \mathscr{M}_i} c_i(u) \tag{2.6}$$

Limitations Shallow embedding techniques have serious limitations rooted at their very design [125], as they learn a vector for each node by directly optimizing the matrix $\mathbf{Z}$, which makes the approach computationally inefficient as there is no parameter-sharing and the number of parameters grows linearly with the number of nodes ($\mathscr{O}(d \cdot |V|)$). This leads to another drawback as the approach is inherently *transductive*, i.e., does not support any kind of generalization to new nodes or networks. Moreover, feature-rich networks are not supported since the encoder only maps nodes to vectors from the matrix, without any way to provide node or edge features.

2.4.2 *Deep* embedding: Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs), often called graph convolutional networks for their similarity with Convolutional Neural Networks, are another class of algorithms that aim to learn on graphs and that is becoming increasingly popular thanks to its properties and performance in many tasks.

The name and the idea of GNNs dates back to 2008 with the seminal work by Scarselli et al. [159], who proposed an information diffusion based model that extended the Recurrent Neural Networks to support generic graphs (i.e., cyclic, directed and undirected, whereas RNNs' input is limited to acyclic directed graphs only), and also extended the random walk

models. Such model produced an output for each node by aggregating the local information (node and edge features, and the output at the previous processing step) from its neighborhood using functions with learnable parameters. After many years being unnoticed, GNNs have been lately re-discovered, and the local-neighborhood aggregation idea at their core, which is somehow similar to the one of CNNs, became the generic framework that inspires most of the works proposed in the literature.

In particular, by Graph Neural Networks we now mean a generic *neural message passing* architecture where each layer of the model performs a message passing iteration. Graph Neural Networks are often considered a generalization of the convolution operator, used by Convolutional Neural Networks (CNNs) and that made the success of Deep Learning, to graph-structured data [127]. In fact, while the CNNs are well-defined for Euclidean data (images are grids of pixels), the convolution needs a new formulation for non-Euclidean domains like graphs and manifolds. Such convolution operator for graphs should be used to define more complex encoder functions that leverage both the topology of the graph and also the node features. Specifically, like in the Euclidean convolutional models where, in each convolutional layer, a kernel (i.e., a learned tensor) is used to multiply and aggregate the neighborhood of each pixel, the graph convolution operator aggregates the features of the local neighbors of each node using some learned function. The output values provided by the aggregation are higher-level features that will be fed to the next layer of the model, producing a hierarchical representation.

In the last years, many GNN layers have been proposed by researchers and the interest in the field keeps growing thanks to the large availability of graph structured data and to the many interesting applications. For instance, GNNs have been used to perform node classification or regression — e.g., classifying entities like users or assigning them a score —, but also to perform graph classification or regression (by aggregating all the node embeddings together) and even to perform link prediction — i.e., the prediction of whether a link (modeling a relation, interaction, etc.) exists between two nodes (e.g., users, molecules, etc.).

In the following, we formalize and detail GNNs, their usage and their applications.

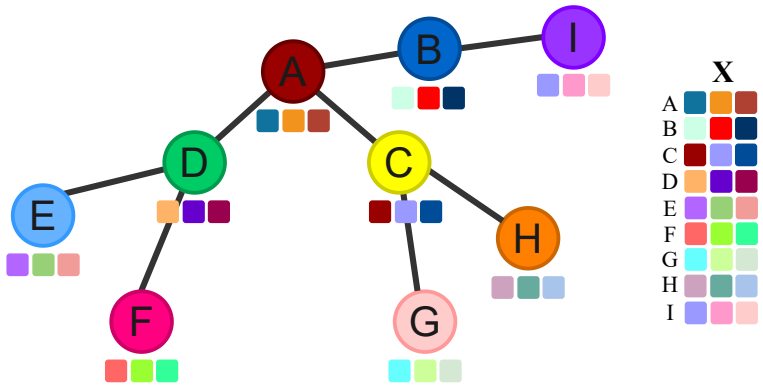
2.4.2.1 The message passing architecture detailed

Formally, the $(k+1)$-th Graph Neural Network layer computes the embedding $\mathbf{h}_u^{(k+1)}$ of node u as [125]:

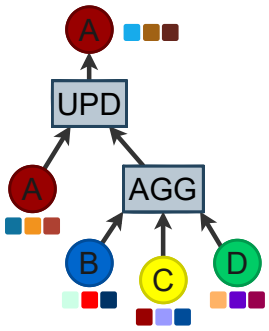
$$\mathbf{h}_u{}^{(0)} = \mathbf{x}_u \tag{2.7}$$

$$\mathbf{h}_u{}^{(k+1)} = \text{UPDATE}^{(k)}(\mathbf{h}_u^{(k)},\ \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}_u)\}) \tag{2.8}$$

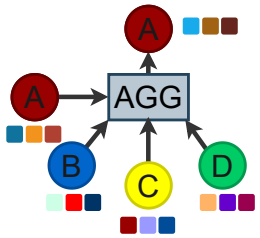
where $\mathbf{x}_u$ are the input node features, $\mathcal{N}_u$ is the local neighborhood of node u, and UPDATE$^{(k)}$ and AGGREGATE$^{(k)}$ are two arbitrary learnable and differentiable functions. In other words, the features of the target node u are updated with a learnable function that accounts for the current features and for the (learnable) aggregation of the ones of its neighbors. As an example, we illustrate a single propagation step in Figure 2.1b.



(a) A toy-example rich-network with the input node features tensor $\mathbf{X}_0$.



(b) Embedding computation.



(c) Embedding computation with self-loops.

Fig. 2.1 **GNN message passing example.** Simple toy-example to show a propagation step performed by a single GNN layer.

It should be noted that this formulation produces node embeddings that depend both on the input node features $\mathbf{x}_u$ and on the topology of the graph.

Graph Neural Networks share many desirable properties with their Euclidean ancestors. For instance, the locality of the convolutional operator ensures low computational complexity; the parameter sharing (i.e., all nodes are processed by the same Neural Networks) makes them spatially efficient and allows them to generalize to previously unseen nodes and graphs (*inductive* capability); they are permutation invariant. In comparison, *shallow* embedding methods lack these properties (plus the leveraging of node features), which explains why the GNN approach has become so popular.

2.4.2.2 Message passing with self-loops

A simplification of the GNN framework in equation 2.8, used by many layers in the literature, consists in removing the UPDATE by considering each node as its own neighbor (i.e., by adding self-loops). The generic GNN layer then becomes [125]:

$$\mathbf{h}_u^{(k+1)} = \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}_u \cup \{u\}\}) \tag{2.9}$$

While equation 2.9 has the advantage of simplifying the training and reduce overfitting [125], it has the main drawback of not being able to distinguish between the incoming features from the neighbors and the node itself. As an example, we illustrate a single propagation step in a GNN with self-loops in Figure 2.1c.

2.4.3 Generic model architecture

So far we have defined the generic Graph Neural Network layer and their advantages over shallow embeddings. Let's see how they are used in a generic Geometric Deep Learning model.

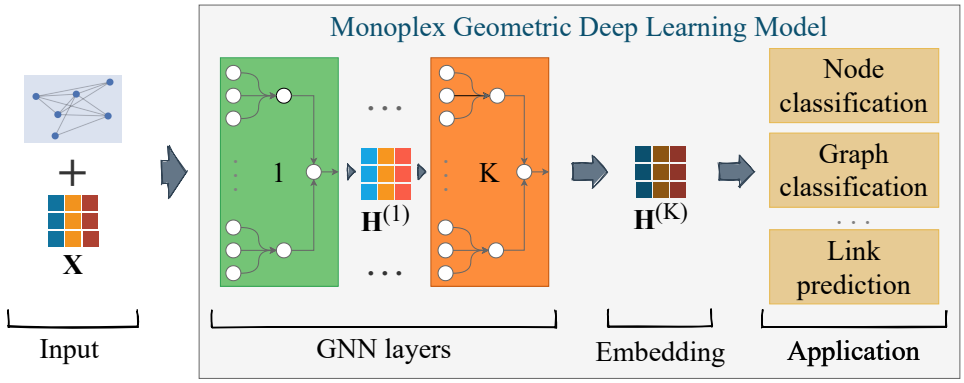


Fig. 2.2 **A Geometric Deep Learning model example.** K is the number of convolutional layers, $\mathbf{X}$ are the input node features. The node embeddings $\mathbf{H}^{(K)}$, can be used in various applications.

As shown in Figure 2.2, the generic Geometric Deep Learning model takes as input the graph plus the features of its nodes ($\mathbf{X}$) and, depending on the GNN layers used, also link weights or features.

The first processing of the graph happens in the GNN layers, where the topology is used to propagate and combine the node (and the edge) features. After K GNN layers, each followed by an activation function, the output node embeddings (that are high-level

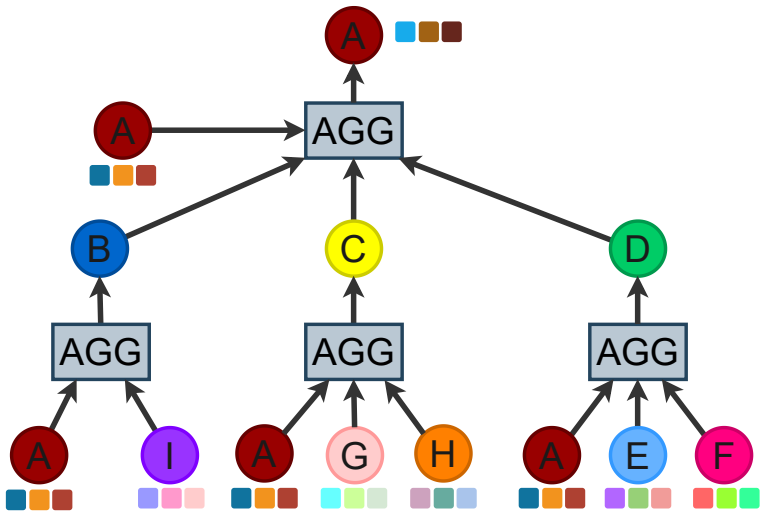


Fig. 2.3 **Example of computational tree** of a model with two GNN layers ($K = 2$) in the network shown in Figure 2.1a. For simplicity's sake, we consider layers without the UPDATE function.

node features learned hierarchically) can be used for any task and fed, for instance, to a Multi-Layer Perceptron to perform node classification or regression.

From a graph perspective, the computation of node u's embedding $\mathbf{h}_u^{(K)}$ is the propagation of the node features in a bottom-up fashion in a tree where the root is the node u itself and the nodes at each layer k are u's k-hops neighbors. After K iterations, $\mathbf{h}_u^{(K)}$ are the (final) node embedding of node u. This means that the more GNN layers are stacked together, the farther the node features will propagate in the graph, building higher level node features hierarchically that capture some high-order pattern; for comparison, in the CNN case lines are combined to find angles, which are combined to find shapes, etc. That is, a model with K GNN layers is able to capture and aggregate the whole K-hop neighborhood of each node u.

As an example, in Figure 2.3 we show the propagation tree of a model with two GNN layers that use self-loops.

2.4.4 Pooling layers

Like in the case of Euclidean convolutional models, *pooling layers* that aggregate the embeddings of nodes have been defined. There are two motivations for aggregating nodes via pooling layers:

- Learning the embedding of the entire graph, as discussed in Section 2.4.5;

- Coarsening the input graph to learn hierarchical representations, like in CNNs. The next GNN layer in the model will take as input a coarsened graph where each node (and its features) is the aggregation of a cluster of nodes.

While the simplest pooling layer consists in considering the *mean* or the *max* of a sub-set of node embeddings, more complex ones have been proposed.

For instance, Li et al. [160] define a graph-level representation that can be generalized as in the following [161]:

$$\mathbf{r}_i = \sum_{n}^{\mathscr{G}_i} \text{softmax}\left(\Theta_{\text{att}}(\mathbf{H}_i^{(K)} \,\|\, \mathbf{X}_i)\right)_n \odot \Theta_{\text{nn}}(\mathbf{h}_n^{(K)} \,\|\, \mathbf{x}_n) \tag{2.10}$$

where $\mathscr{G}_i$ is either the entire graph or a sub-graph, $\mathbf{H}_i$ and $\mathbf{X}_i$ are embedding of and the input features of $\mathscr{G}_i$, $\Theta_{\text{att}}\colon \mathbb{R}^F \to \mathbb{R}$ is a neural network that provides a soft-attention score for the node and decides what nodes are relevant for the task, while $\Theta_{\text{nn}}\colon \mathbb{R}^F \to \mathbb{R}^{F'}$ is a generic neural network.

In another work, Ying et al. propose *DiffPool* [162] to coarsen the graph with learned cluster assignments. Specifically, they aggregate the graph as in the following:

$$\mathbf{H}^{(k+1)} = \text{softmax}(\mathbf{S}^{(k)})^\top \cdot \mathbf{H}^{(k)} \tag{2.11}$$

$$\mathbf{A}^{(k+1)} = \text{softmax}(\mathbf{S}^{(k)})^\top \cdot \mathbf{A}^{(k)} \cdot \text{softmax}(\mathbf{S}^{(k)}) \tag{2.12}$$

where $\mathbf{A}$ is the adjacency matrix, $\mathbf{H}$ is the node embedding matrix and $\mathbf{S} \in \mathbb{R}^{N^{(k)} \times C^{(k+1)}}$ is the learned cluster assignment. Note that $N^{(k)}$ is the number of nodes after layer k and $C^{(k+1)}$ is the number of clusters, and that each $a_{ij} \in \mathbf{A}^{(k+1)}$ represents the connectivity strength between clusters i and j. Regarding $\mathbf{S}$, it is learned by adding the following two auxiliary loss functions, meant to make the training of the coarsening matrix easier, to the loss of the task.

$$\mathscr{L}_{LP}^{(k)} = \|\mathbf{A}^{(k)} - \text{softmax}(\mathbf{S}^{(k)})\ \text{softmax}(\mathbf{S}^{(k)})^\top\|_F \tag{2.13}$$

$$\mathscr{L}_{E}^{(k)} = \frac{1}{N^{(k)}} \sum_{n=1}^{N^{(k)}} H(\mathbf{S}_n^{(k)}) \tag{2.14}$$

where H is the entropy function. While $\mathscr{L}_{LP}^{(k)}$ is the link prediction loss, $\mathscr{L}_{E}^{(k)}$ is used to regularize the entropy of the clustering.

2.4.5 Graph embedding

The GNN framework discussed so far does not allow to compute a *graph embedding* directly. However, many applications require graph-level information, for instance in the case of graph classification or regression. The common approach to overcome this limitation is to aggregate the embeddings of all the nodes in the graph via *pooling layers*. The motivation is that, since the node embeddings capture some local characteristics of the graph, by aggregating them one would get some global information about the graph itself.

2.4.6 Link embedding

The GNN framework does not provide an explicit way to compute the embedding of links. The common approach in the literature is to apply some function to the embeddings of the end nodes, like concatenation, sum, element-wise product or even a Multi-Layer Perceptron. This can be motivated from a Network Science perspective with the fact that the link is a relation between the two nodes, and as such it directly depends on the nodes themselves (and, therefore, on their embedding).

As an example, if concatenation is applied, the embedding of the link e_{uv} from node u to node v is $\mathbf{h}_{(u,v)}^{(K)} = \mathbf{h}_u^{(K)} \,||\, \mathbf{h}_v^{(K)}$.

2.4.7 Some examples of GNN layers

After defining the general framework, let's have a look to some of the most common GNN layers.

2.4.7.1 Graph Convolutional Networks (GCN)

One of the first layers defined after the re-emergence of GNNs is Graph Convolutional Networks (*GCN*) by Kipf et al. [163]. GCNs use self-loops to remove the UPDATE function, and only have one trainable matrix. The incoming embeddings from each neighboring node is normalized by the (square root of the) product of the degrees (strengths) of the two nodes.

Formally, the GCN layer is defined as:

$$\mathbf{h}_u^{(k+1)} = \mathbf{W}^{(k)} \sum_{\forall v \in \mathcal{N}_u \cup \{u\}} \frac{e_{v,u}}{\sqrt{s_u s_v}} \mathbf{h}_v^{(k)} \tag{2.15}$$

where $\mathbf{W}^{(k)}$ is the trainable weight matrix, $e_{v,u}$ is the weight of the link from v to u (1 if the graph is unweighted), and s_v and s_u are the strengths of the nodes.

Equation. 2.15 can also be defined in matrix form as follows:

$$\mathbf{H}^{(k+1)} = \left(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\right)\mathbf{H}^{(k)}\mathbf{W}^{(k)} \tag{2.16}$$

where $\mathbf{H}$ is the features' tensor, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with self-loops and $\hat{\mathbf{D}}$ is the diagonal degree matrix of $\hat{\mathbf{A}}$. Using $\hat{\mathbf{A}}$ instead of $\mathbf{A}$ is known as *renormalization trick* and solves numerical instabilities and exploding/vanishing gradients in deep models.

Thus, time complexity of a GCN layer is:

$$O(\text{GCN}) \simeq O(f^{(k)} \cdot f^{(k+1)} \cdot |V| + f^{(k+1)} \cdot |E|) \tag{2.17}$$

where $f^{(k)}$ is the number of (input) features at layer k and $f^{(k+1)}$ is the number of output features, and $|V|$ and $|E|$ are the number of nodes and edges respectively.

2.4.7.2 GraphSAGE

In [130], Hamilton et al. propose *GraphSAGE* to process large graphs, thanks to the sampling performed in each node neighborhood.

Formally, *GraphSAGE* layers are defined as:

$$\mathbf{h}_u^{(k+1)} = \mathbf{W}^{(k)} \cdot [\mathbf{h}_u^{(k)} \,\|\, \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}_u^K\})] \tag{2.18}$$

where $\mathcal{N}_u^K$ is the sampled neighborhood of node u and includes K nodes.

The aggregation function should be symmetric (i.e. invariant to input permutations), which ensures that the model can be trained and applied to any isomorphism and produce the same result. The authors define three variants based on three different aggregation functions:

- Mean aggregator, very similar to the base and GCN approach, it just computes the mean between the features;
- Pooling aggregator, which returns the element-wise minimum or maximum;
- RNN aggregator, which uses a Long-Short Term Memory (LSTM) with randomly permuted inputs, since LSTMs are non-symmetric per se.

The time complexity of a GraphSAGE layer is:

$$O(\text{GraphSAGE}) \simeq O(r^{(k)} \cdot f^{(k)} \cdot f^{(k+1)} \cdot |V|) \tag{2.19}$$

where $f^{(k)}$ is the number of (input) features at layer k, $f^{(k+1)}$ is the number of output features, $|V|$ is the number of nodes and $r^{(k)}$ is the number of sampled neighbors for each node.

2.4.7.3 Graph Attention Networks (GAT)

An important work by Veličković et al. [164] borrows the *attention mechanism* [165] from the Natural Language Processing (NLP) field to assign importance weights to each neighboring node and improve the aggregation function. Specifically, they compute an *attention* coefficient $\alpha_{u,v}$ for each neighboring node, used to scale its incoming embedding. This definition resembles the one of *transformers* used in NLP, the main difference being that the attention is computed only for the neighboring nodes and not for the full graph.

GAT do not define an UPDATE function, and the AGGREGATE is just the sum of the features of its neighbors, scaled with their own attention coefficient.

Formally, the GAT layer is defined as:

$$\mathbf{h}_u^{(k+1)} = \mathbf{W}^{(k)} \sum_{\forall v \in \mathcal{N}_u \cup \{u\}} \alpha_{u,v}^k \mathbf{W}^{(k)} \mathbf{h}_v^{(k)} \tag{2.20}$$

where $\mathbf{W}^{(k)}$ is the trainable parameters matrix,

$$\alpha_{u,v}^{(k)} = \text{softmax}(\text{LeakyReLU}(\mathbf{e}_u^{(k)}))_v \tag{2.21}$$

and

$$e_{u,v}^{(k)} = \mathbf{a}^\top [\mathbf{W}^{(k)} \mathbf{h}_u^{(k)} || \mathbf{W}^{(k)} \mathbf{h}_v^{(k)}] \tag{2.22}$$

Graph Attention Networks also allow defining multiple attention *heads*. That is, each layer can compute the embedding multiple times (one per head, each with their own independent weights matrix) and then concatenate or sum them. This is meant to capture multiple patterns, in a similar way to the multiple filters (3D kernels) applied in the CNNs.

The time complexity of a GAT layer is

$$\mathcal{O}(\text{GAT}) \simeq \mathcal{O}(h(f^{(k)} \cdot f^{(k+1)} \cdot |V| + f^{(k+1)} \cdot |E|)) \tag{2.23}$$

where $f^{(k)}$ is the number of (input) features at layer k and $f^{(k+1)}$ is the number of output features, h is the number of *heads*, and $|V|$ and $|E|$ are the number of nodes and links in the network.

2.4.7.4 Simple Graph Convolution (SGC)

In their work, Li et al [166] show that GCNs, and in general convolutional-style layers, are affected by the *over-smoothing* of the hidden representations of locally connected nodes. Specifically, due to the feature propagation in the K-hops neighborhood, deep models tend to

lose the local neighborhood information and to produce similar embeddings for all nodes. This is particularly true for layers that use the self-loop update mechanism, but also affects any model where the weight assigned the neighbors' features is larger than the one of the node itself.

Wu et al. [167] address the problem of *over-smoothing* and try to relieve its effect by defining the *Simple Graph Convolution* layer (SGC), a "possible predecessor" of GCNs obtained removing most of their complexity. More in detail, they remove the non-linearity between the GCN layers and collapse the resulting function in a single linear transformation. In their experiments, the resulting model achieves the same or superior performance to the GCNs in many tasks.

In matrix form, SGC can be expressed as

$$\mathbf{Y} = \left(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\right)^{K}\mathbf{X}\mathbf{W}^{(K)} \tag{2.24}$$

where $\mathbf{Y}$ is the output of the model, $\mathbf{X}$ is the node features matrix, $\mathbf{W}^{(k)}$ is the trainable parameters matrix, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with self-loops and $\hat{\mathbf{D}}$ is the diagonal degree matrix of $\hat{\mathbf{A}}$. As in GNNs in equation 2.16, the renormalization trick is used. However, the main difference with GNNs is that SGC process the K-hop neighborhood directly in a single layer, producing the output for each from the input matrices. From this perspective, eq. 2.24 can be re-written as:

$$\mathbf{S} = \left(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\right) \tag{2.25}$$

$$\mathbf{Y} = \mathbf{S}^{K}\mathbf{X}\,\mathbf{W}^{(K)} \tag{2.26}$$

. The authors note that the first product of the equation corresponds to the feature extraction phase, while the second corresponds to applying a fully connected layer, e.g., for classification or regression tasks. In other word, the SGC do not perform message passing, which solves the over-smoothing problem and improves the time complexity of the model.

2.4.7.5 GCN via Initial residual and Identity mapping (GCNII)

In their work, Chen et al. [168] propose Graph Convolutional Network via Initial residual and Identity mapping (GCNII), a GNN layer that extends the Graph Convolutional Networks (GCN) and addresses the over-smoothing problem in a different way than the SGC discussed previously, the final goal being allowing the use of deep networks. Specifically, they propose a layer with a residual connection to (a learned function of) the input node features and with

an identity mapping. Both have various empirical and theoretical motivations, and we refer the Reader to their paper for the details.

Formally, the GCNII layers are defined as:

$$\mathbf{H}^{(k+1)} = \left((1-\alpha)\mathbf{S}\mathbf{H}^{(k)} + \alpha\mathbf{H}^{(0)}\right)\left((1-\beta)\mathbf{I} + \beta\mathbf{W}^{(k)}\right) \tag{2.27}$$

$$\beta = \log\left(\frac{\theta}{k}+1\right) \tag{2.28}$$

where $\mathbf{H}^{(k)}$ is the node embeddings in matrix form, $\mathbf{H}^{(0)}$ is the (input) node features matrix or its learned transformation (i.e., via fully-connected layers) to reduce/augment the number of dimensions, $\mathbf{W}^{(k)}$ is the trainable parameters matrix, $\mathbf{S} = \left(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\right)$, $\hat{\mathbf{A}} = \mathbf{A}+\mathbf{I}$ is the adjacency matrix with self-loops and $\hat{\mathbf{D}}$ is its diagonal degree matrix (the renormalization trick is used), α is the strength of the residual connection to the input node features and θ is a hyperparameter used to compute the strength of the identity mapping. It should be noted that, the deeper the layer (i.e., the higher k), the smaller the identity mapping will be.

The computational complexity of GCNII is the same as GCNs, i.e., it is:

$$O(\text{GCNII}) \simeq O(f^{(k)} \cdot f^{(k+1)} \cdot |V| + f^{(k+1)} \cdot |E|) \tag{2.29}$$

where $f^{(k)}$ is the number of (input) features at layer k and $f^{(k+1)}$ is the number of output features, and $|V|$ and $|E|$ are the number of nodes and edges respectively.

2.4.7.6 SignedGCN

SignedGCN (SGCN) [169], proposed by Derr et al., is a GNN layer designed to process signed networks and has motivations based on balance theory. In particular, each SGCN layer produces two representations for each node, one for the positive and one for the negative neighbors. Formally, the first SGCN layer in a model with K layers is defined as [161]

$$\begin{aligned}
\mathbf{h}^{(1)}_{u_{\text{pos}}} &= \mathbf{W}^{(0)}_{\text{pos}}\left[\frac{1}{|\mathcal{N}_u^+|}\sum_{v\in\mathcal{N}_u^+}\mathbf{x}_v, \mathbf{x}_u\right] \\
\mathbf{h}^{(1)}_{u_{\text{neg}}} &= \mathbf{W}^{(0)}_{\text{neg}}\left[\frac{1}{|\mathcal{N}_u^-|}\sum_{v\in\mathcal{N}_u^-}\mathbf{x}_v, \mathbf{x}_u\right]
\end{aligned} \tag{2.30}$$

and the following ones as [161]

$$\begin{aligned}
\mathbf{h}_{u_{\text{pos}}}^{(k+1)} &= \mathbf{W}_{\text{pos}}^{(k)} \left[\frac{1}{|\mathcal{N}_u^+|} \sum_{v \in \mathcal{N}_u^+} \mathbf{h}_{v_{\text{pos}}}^{(k)}, \frac{1}{|\mathcal{N}_u^-|} \sum_{v \in \mathcal{N}_u^-} \mathbf{h}_{v_{\text{neg}}}^{(k)}, \mathbf{h}_{u_{\text{pos}}}^{(k)} \right] \\
\mathbf{h}_{u_{\text{neg}}}^{(k+1)} &= \mathbf{W}_{\text{neg}}^{(k)} \left[\frac{1}{|\mathcal{N}_u^+|} \sum_{v \in \mathcal{N}_u^+} \mathbf{h}_{v_{\text{neg}}}^{(k)}, \frac{1}{|\mathcal{N}_u^-|} \sum_{v \in \mathcal{N}_u^-} \mathbf{h}_{v_{\text{pos}}}^{(k)}, \mathbf{h}_{u_{\text{neg}}}^{(k)} \right]
\end{aligned} \tag{2.31}$$

where $\mathbf{h}_{u_{\text{pos}}}^{(k)}$ and $\mathbf{h}_{u_{\text{neg}}}^{(k)}$ are respectively the positive and negative representations of node u after k SGCN layers, and $\mathcal{N}_u^+$ and $\mathcal{N}_u^-$ are the neighbors of node u connected by positive and negative links respectively. The output embedding $\mathbf{h}_u^{(K)}$ is then computed as the concatenation of the positive and negative representations:

$$\mathbf{h}_u^{(K)} = \mathbf{h}_{u_{\text{pos}}}^{(K)} \,\|\, \mathbf{h}_{u_{\text{neg}}}^{(K)} \tag{2.32}$$

2.4.7.7 Temporal Graph Neural Networks (TGN)

The Graph Neural Network layers analyzed so far do not handle any temporal information explicitly. However, real-world graphs are, in many cases, the result of a temporal dynamics. For instance, acquaintance or friendship networks may gain new links when users meet new people, or may even lose some of them in time. Thus, if the temporal dynamics has a role in the target of the learning, ignoring it and learning on the static graph instead may lead to wrong results. To overcome this limitation, some time-aware GNNs have been proposed. While most attempts focus on Discrete-Time Dynamic Graphs (DTDG), i.e., the temporal dynamics of the graph is represented by a list of graph snapshots, *Temporal Graph Neural networks* (TGN) have been recently proposed by Rossi et al. [147] to deal with Continuous-Time Dynamic Graphs (CTDG), i.e., the timestamp of the events is continuous. That is, they can handle graphs where the timestamp events are not binned, and produce the node embeddings at each timestamp. Specifically, TGN support node-wise and interaction events (edges between source and target nodes), and also the addition and deletion of nodes and interactions (edges).

How do TGN work? The key idea of Rossi et al. is to include a *memory module* that stores the state for each node and that should encode the node's history and long-term dependencies. Specifically, when new nodes are seen for the first time, their memory entry is created and initialized to a vector of zeros. The state of each node is then updated after each event involving that node and is fed along with the input node features when node embeddings are produced.

In order to process events, authors first transform them into messages using some (learnable) function that produces a *message* for each node involved. For instance, a node-wise event is transformed into a single message for the node, whereas interaction-events are transformed in a message for the source and one for the target nodes. Each event will be later used to update the memory. For sake of simplicity, in their work they choose the identity function to transform events into messages (i.e., no learning is performed at this stage). It is worth mentioning that, during the training, Rossi et al. divide the events in batches to improve the efficiency. In each batch the messages involving the same node are aggregated using a *message aggregator* (e.g., by mean or by keeping only the most recent).

Regarding the embedding generation, they propose various embedding strategies. The most interesting employs a slightly modified *Temporal Graph Attention* (TGAT) from Xu [170], an extension of the Graph Attention Networks (GAT) for Discrete-Time Dynamic Graphs. In particular, it generates embeddings at a generic timestamp t as follows:

$$\mathbf{h}_u^{(k)}(t) = \text{MLP}^{(k)}(\mathbf{h}_u^{(k-1)}(t), \tilde{\mathbf{h}}_u^{(k)}(t)) \tag{2.33}$$

$$\tilde{\mathbf{h}}_u^{(k)}(t) = \text{MultiHeadAttention}^{(k)}(\mathbf{q}^{(k)}(t), \mathbf{K}^{(k)}(t), \mathbf{V}^{(k)}(t)) \tag{2.34}$$

$$\mathbf{q}^{(k)}(t) = \mathbf{h}_u^{(k-1)}(t) \,\|\, \phi(0) \tag{2.35}$$

$$\mathbf{K}^{(k)}(t) = \mathbf{V}^{(k)}(t) = \mathbf{C}^{(k)}(t) \tag{2.36}$$

$$\mathbf{C}^{(k)}(t) = \Big\|_{v=1}^{N} \Big[\mathbf{h}_v^{(k-1)}(t) \,\|\, \mathbf{e}_{uv}(t_v) \,\|\, \phi(t - t_v)\Big] \tag{2.37}$$

$$\mathbf{h}_u^{(0)}(t) = \mathbf{s}_u(t) + \mathbf{v}_u(t) \tag{2.38}$$

where ϕ is a generic time encoding, $\mathbf{s}_u(t)$ and $\mathbf{v}_u(t)$ are the memory entry and the features of node u, and where each attention head of equation. 2.34 is computed as in [165]:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.39}$$

with Q, K and V being the query node, the keys and the values respectively and d_k being the number of dimensions of the keys K.

It should be noted that input node features in equation 2.38 are sum with the memory entry of that node (i.e., the embedding of its history), so the spatial convolution also accounts for the past events. Using a spatial convolutional network to generate the embeddings has the advantage of countering the *stale memory* problem that occurs when the memory of a node has not been updated in a long time. In fact, even if there is no recent event regarding that

node (i.e., its memory has not been updated recently), it is likely that one of its neighbors has been involved in some event. Moreover, as shown in equation 2.37, each message-passing iteration in the convolutional network also includes information about the edge (the event) that links the two nodes, and its encoded time.

Authors test TGN in the node classification and link prediction tasks on various datasets, including Twitter networks with temporal information about tweets, retweets, etc. and, according to their results, they outperform any other static or DTDG convolutional network tested.

For more information about the TGN and the TGAT, we refer the Reader to the respective papers.

2.4.8 Training methodology

While the training methodologies of Geometric Deep Learning are the same of the Euclidean counterpart (e.g., supervised, unsupervised, reinforcement), due to the graph-structure of the data, a more precise definition of the generalization capabilities of a model to new data is required. In fact, while in the case of Euclidean data the new observations are independent of the others, in the case of nodes (or links) there is a relationship with the other observations (e.g., a new node may be connected to others, a link connects two nodes, etc.).

To clarify the generalization capabilities of the model, the following terms are used to define the methods [125]:

- *Transductive*: a model that is trained on the full graph, without computing the loss on test nodes (i.e., without providing their label), and that is used to predict the label of the test nodes. This implies that the entire structure of the graph is leveraged during training, affecting also the embedding (thus, the predictions) on test nodes. Considering that labeled and unlabeled data is used together during the training, the training methodology is often called *semi-supervised*;

- *Inductive*: a model trained on a sub-graph without the test nodes and their edges, and will later predict their labels on the full graph. That is, test nodes are completely unobserved during training. The training methodology is, therefore, called *supervised*.

2.5 Explaining the GNNs

The good performance of Graph Neural Networks in many interesting and meaningful applications gave rise to the following question: "*why do GNN models work?*" or, more

specifically: "*What characteristics of the graph do they capture? And how?*". Answering these questions would not only open a window on these black-boxes, making them more transparent, but would also offer insights about the problem resolution, that could help better understand the problem itself. That is the case, for instance, of medical applications where it might be hard to find a common pattern between patients by human inspection. Moreover, understanding the GNNs could help in fixing the mistakes the model makes by improving the model itself or with new data.

2.5.1 GNNExplainer

The first and most famous attempt to answer these questions is *GNNExplainer* [171].

Given the prediction on a node, GNNExplainer returns a sub-graph $\mathcal{G}_s$ of the input graph $\mathcal{G}$ and the weight of each input feature such that the prediction error (w.r.t. the original prediction) made on the sub-graph and the weighted features is reasonably small.

An advantage of GNNExplainer is that it considers the model as a black-box: it is model agnostic as it acts only on the input graph and features. However, the initial formulation is only suitable for classification tasks.

Formally, for each node, the sub-graph and the features mask are learned by maximizing the mutual information following equation:

$$\max_{\mathcal{G}_s,F} \mathrm{MI}(Y,(\mathcal{G}_s,\mathbf{X}_s^F)) = H(Y) - H(Y|\mathcal{G}=\mathcal{G}_s,\mathbf{X}=\mathbf{X}_s^F) \tag{2.40}$$

which is equivalent to minimizing the following conditional entropy, as the entropy is constant for a trained model (i.e., predictions do not change):

$$\min_{\mathcal{G}_s,F} H(Y|\mathcal{G}=\mathcal{G}_s,\mathbf{X}=\mathbf{X}_s^F) \tag{2.41}$$

where Y is the probability of the node belonging to each of the classes, $\mathcal{G}$ is the input graph and $\mathcal{G}_s$ its explanation sub-graph, $\mathbf{X}$ is the matrix of input node features and $\mathbf{X}_s^F$ the matrix of the node features masked by F. That is, GNNExplainer attempts to learn what most affected the prediction for a given node by maximizing the probability of the prediction, and does so by minimizing the uncertainty of the model when the prediction is limited to $\mathcal{G}_s$ with features matrix $\mathbf{X}_s^F$. Note that $\mathcal{G}_s \subseteq \mathcal{G}_c$, the computation sub-graph for that node, i.e., that sub-graph of the node's K-hop neighborhood in which a model with K GNN layers propagates the features.

It is also worth mentioning that the above optimization is not directly tractable as the possible number of G_s is exponential with the number of nodes in $\mathscr{G}_c$, and the authors learn an approximation that works well in practice.

While the computational complexity of GNNExplainer depends on the computation graph of each node, authors note that the resulting explanation sub-graphs are connected and smaller than the K-hop neighborhoods in which a model with K GNN layers propagates the features, which allows GNNExplainer to operate even on large graphs.

Authors also propose a way to perform class-level explanations. For more details, we refer the Reader to their paper.

2.6 Software libraries

One of the main issues of developing applications based on cutting-edge algorithms and techniques is the lack of high quality implementations. However, that is not the case of Geometric Deep Learning. In fact, despite it being a recent field, there are many frameworks that provide stable and high performance implementations of the most popular algorithms in the literature.

While the most popular (according to the stars and forks of the respective GitHub repositories) is *PyTorch Geometric* [161], an open-source library built on-top of PyTorch [172] that is actively developed and receives large community contributions, other libraries exist and are gaining popularity. For instance, Deepmind released *Graph Nets* [173] in 2018 and later *Jraph* [174] for JAX. Other libraries are *Stellargraph* [175] and Deep Graph Library (DGL) [176].

Chapter 3

Learning Network Dismantling

3.1 Introduction

Several empirical systems consist of nonlinearly interacting units, whose structure and dynamics can be suitably represented by complex networks [177]. Heterogeneous connectivity [178], mesoscale [179, 180], higher-order [181, 182] and hierarchical [183] organization, efficiency in information exchange [184] and multiplexity [80, 81, 73, 87], are distinctive features of biological molecules within the cell [185], connectomes [186], mutualistic interactions among species [187], urban [188], trade [189] and social [190–192] systems.

However, the structure of complex networks can dramatically affect its proper functioning, with crucial effects on collective behavior and phenomena such as synchronization in populations of coupled oscillators [193], the spreading of infectious diseases [194, 195] and cascade failures [196], the emergence of misinformation [197, 198] and hate [199] in socio-technical systems or the emergence of social conventions [200]. While heterogeneous connectivity is known to make such complex networks more sensitive to shocks and other perturbations occurring to hubs [71], a clear understanding of the topological factors — and their interplay — responsible for a system's vulnerability still remains elusive. For this reason, the identification of the minimum set of units to target for driving a system towards its collapse — a procedure known as network dismantling — attracted increasing attention [201–205] for practical applications and their implications for policy-making. Dismantling is efficient if such a set is small and, simultaneously, the system quickly breaks down into smaller isolated clusters. The problem is, however, NP-hard and while percolation theory provides the tools to understand large-scale transitions as units are randomly disconnected [206, 207, 74, 208], a general theory of network dismantling is missing and applications mostly rely on approximated theories or heuristics.

Here, we develop a computationally efficient framework — named GDM (*Graph Dismantling with Machine learning*) and conceptually described in Figure 3.1 – based on machine learning, to provide a scalable solution, tackle the dismantling challenge, and gain new insights about the latent features of the topological organization of complex networks. Specifically, we employ Graph Neural Networks, overcoming the limitations of classic (Euclidean) deep learning and operate on graph-structured data. These layers, inspired by the convolutional layers that empower most of the deep learning models nowadays, aggregate the features of each node with the ones found in its neighborhood by means of a learned non-trivial function, producing high-level node features. While the machine is trained on identifying the critical point from dismantling of relatively small systems — that can be easily and optimally dismantled — we show that it exhibits remarkable inductive capabilities, being able to generalize to previously unseen nodes and way larger networks after the learning phase.

This work follows and combines two recent trends in Machine Learning: learning on synthetic data and generalizing to real-world instances [209], and learning heuristics to tackle/solve hard combinatorial problems on graphs [210, 13]. While the motivation behind the latter is easy to understand, as — thanks to the increasing availability of data — graphs are becoming larger and larger and many interesting applications would be unfeasible due to computational constraints, the idea of learning on synthetic data can be motivated by the unlimited availability of (easily) generated examples with training labels. Thanks to their inductive capabilities and extensive training, Deep Learning models trained on synthetic data are able to generalize to real-world instances, providing a useful tool to approach hard problems in general.

3.2 Proposed framework

3.2.1 Model architecture

The machine learning framework proposed here consists of a (geometric) deep learning model, composed of graph convolutional-style layers and a regressor (a multilayer perceptron), that is trained to predict attack strategies on small synthetic networks – that can be easily and optimally dismantled – and then used to dismantle large networks, for which the optimal solution cannot be found in reasonable time. To give an insight, the graph convolutional-style layers aggregate the features of each node with the ones found in its neighborhood by means of a learned non-trivial function, as they are inspired by the convolutional layers that empower most of the (Euclidean) deep learning models nowadays. More

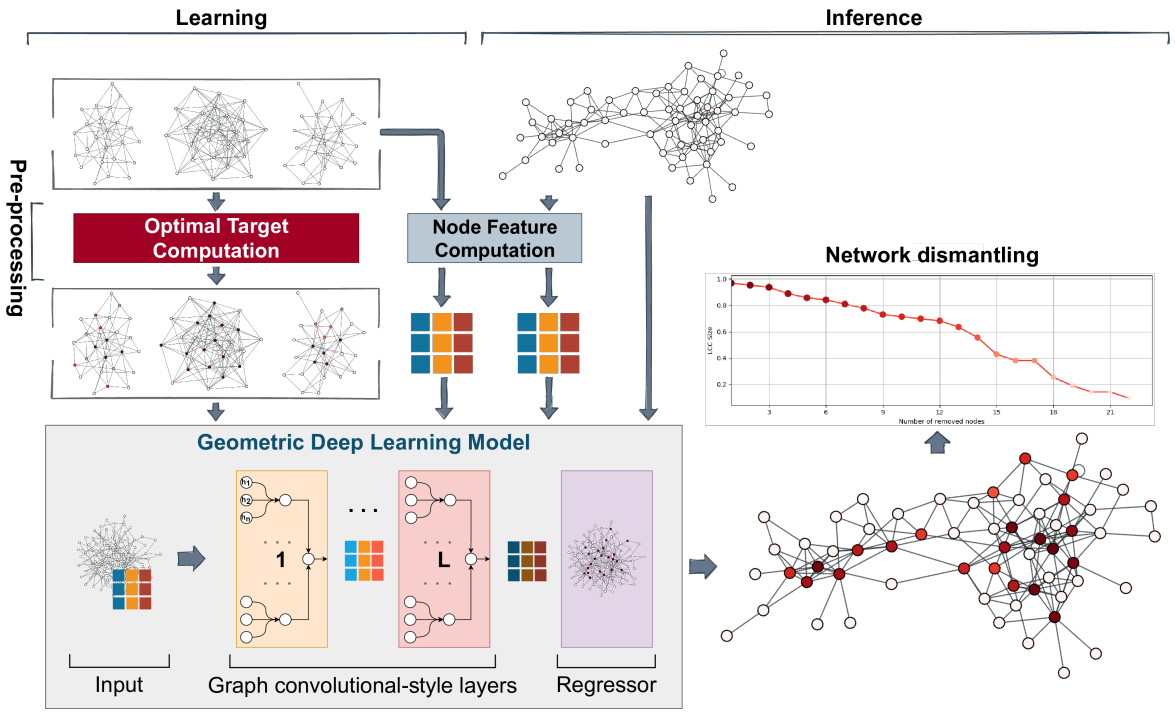


Fig. 3.1 **Training a machine to learn complex topological patterns for network dismantling.** To build our training data, we generate and dismantle small networks optimally and compute the node features. After the model is trained, it can be fed the target network (again, with its nodes' features) and it will assign each node n a value p_n, the probability that it belongs to the (sub-)optimal dismantling set. Nodes are then ranked and removed until the dismantling target is reached. The machine learning architecture used consists of graph convolutional-style layers (*Graph Attention Network* layers) coupled with linear layers — that provide residual connections between consecutive layers — followed by a regressor (i.e., a Multilayer Perceptron) with a sigmoid activation function that constrains the p_n value to the $[0, 1]$ range.

practically, the (higher-order) node features are propagated by the neural network when many layers are stacked: deeper the architecture, i.e., the more convolutional layers, the farther the features propagate, capturing the importance of the neighborhood of each node.

Specifically, we stack a variable number of state-of-the-art layers, namely *Graph Attention Networks (GAT)* [164], discussed in Section 2.4.7.3, that are based on the self-attention mechanism (also known as intra-attention) which was shown to improve the performance in natural language processing tasks [211]. These layers are able to handle the whole neighborhood of nodes without any sampling, which is one of the major limitations of other popular convolutional-style layers (e.g., *GraphSage* [130]), and also to assign a relative importance factor to the features of each neighboring node that depends on the node itself thanks to the attention mechanism. The basic idea is somehow similar to the *Collective Influence* approach, with the main differences being that the geometric deep learning model learns a weighted sum function from the training data to aggregate many node features, whereas the *Collective Influence* just sums the degrees, and also that the model aggregates the whole L-hop neighborhood ball, not just its frontier. These high-level features, h_n^L, are then fed to a regressor that returns p_n, a scalar value between zero and one that represents the node's structural importance indicator used in our work.

The actual implementation of our model relies on *PyTorch Geometric* library [161] on-top of *PyTorch* [212], while the handling of the graphs (i.e., implementation of the data structures, removal of the nodes and the computation of the connected components) is performed using *graph-tool* [213].

3.2.2 Training

We train our models in a supervised manner. Our training data is composed of small synthetic networks (25 nodes each) generated using the Barabási-Albert (BA), the Erdős-Rényi (ER) and the Static Power law generational models that are implemented in *iGraph* [214] and *NetworkX* [215]. Each synthetic network is dismantled optimally using brute-force and nodes are assigned a numeric label (the learning target) that depends on their presence in the optimal dismantling set(s). That is, we find all the minimum size solutions using brute-force (i.e., we try all the combinations of nodes) that reduce the Largest Connected Component (LCC) to a given target size, $\sim 18\%$ in our tests; then, the label of each node is computed as the number of optimal sets it belongs to, divided by the total number of optimal solutions. For example, if there is only a set of optimal size, we assign a label value of 1 to the nodes in that set and 0 to all other nodes; if there are two optimal solutions, we assign 1 to the nodes that belong to both sets, 0.5 to the ones that belong to a single set and 0 to all the others. This

is meant to teach the model that some nodes are more critical than others since they belong to many optimal dismantling sets.

We stress that the training label is arbitrary and others may work better for other training sets or targets. Moreover, while we train on a generic purpose dataset that includes both power law and ER networks, the training networks can also be chosen to fit the target networks, e.g., by using networks from similar domains or with similar characteristics.

3.2.3 Node features

Considering that the model can process any features' combination, one could just choose to stuff every suitable node metrics that comes to their mind and, since it is proven that Deep Neural Networks learn the feature importance, let them do the rest. On the other hand, it could also be tempting to use no features at all (e.g, a constant value for every node) since Kipf et al. [163] showed that their *Graph Convolutional Network (GCN)*, a particular type of convolutional-style graph neural networks, can learn to linearly separate the communities based on the network structure alone and on minimal supervision (one labelled node per community), meaning that convolutional-style neural networks can leverage the network topology to assign a higher-level node feature that describes its role in the network.

We argue that, while the first idea could make sense for scenarios where training data is abundant and the features are cheap to compute, and while the second shows worse (with respect to models with simple features) but still interesting performance, it makes sense to perform some feature selection a priori to keep the computational complexity of the attack low and also to speed up the learning process. With that in mind, we pick node degree (plus its *chi-square value*[1] over the local neighborhood), k–coreness and local clustering coefficient as node features.

3.2.4 Parameters

We run a grid search to test various combination of model parameters, which are reported here, and select the models that better fit the dismantling target (i.e., lower area under the curve or lower number of removals).

- Convolutional-style layers: *Graph Attention Network* layers.
  - Number of layers: from 1 to 4;

[1] The chi-square value of the degree of node i is computed as $\chi_i^2 = (\mathbb{E}[d] - \sigma_d)^2 / \mathbb{E}[d]$, where d is the degree of neighboring nodes.

  - Output channels for each layer: $5, 10, 20, 30, 40$ or 50, sometimes with a decreasing value between consecutive layers;
  - Multi-head attentions: $1, 5, 10, 15, 20$ or 30 concatenated heads;
  - Dropout probability: fixed to 0.3;
  - Leaky ReLU angle of the negative slope: fixed to 0.2;
  - Each layer learns an additive bias;
  - Each layer is coupled with a linear layer with the same number of input and output channels;
  - Activation function: Exponential Linear Unit (ELU). The input at each convolutional layer is the sum between the output of the GAT and the linear layers;
- Regressor: Multi Layer Perceptron
  - Number of layers: from 1 to 4;
  - Number of neurons per layer: $20, 30, 40, 50$ or 100, sometimes with a decreasing value between consecutive layers.
- Learning rate: fixed to 10^{-5};
- Epochs: we train each model for 50 epochs;

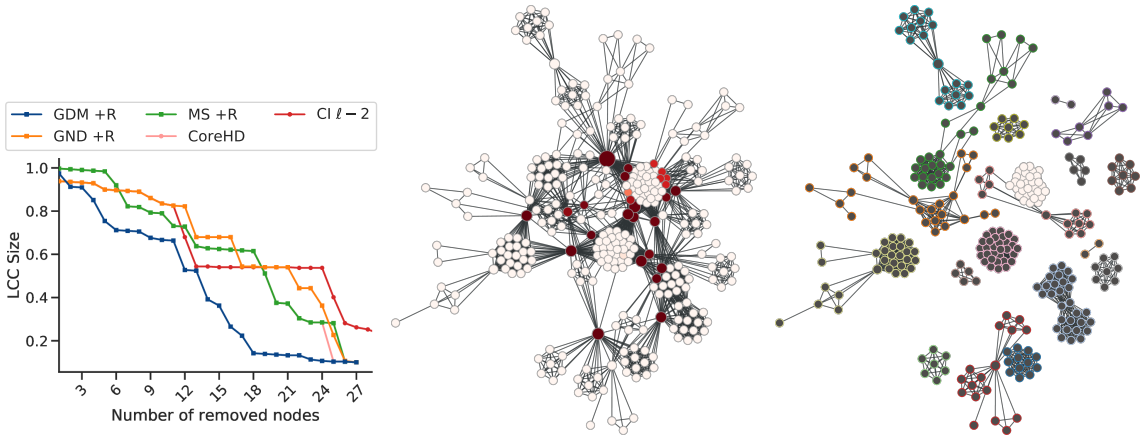
3.3 Dismantling synthetic and real-world systems

In our experiments, we dismantle empirical complex systems of high societal or strategic relevance (e.g., biological, social, infrastructure, communication, trophic and technological systems), our main goal being to learn an efficient attack strategy. To validate the goodness of such a strategy, we compare against state-of-the-art dismantling methods, such as *Generalized Network Dismantling (GND)* [205], *Explosive Immunization (EI)* [216], *CoreHD* [217], *MinSum (MS)* [204] and *Collective Influence* (CI) [203], using local (node degree and its χ^2 value), second-order (local clustering coefficient), and global (k–core value) node features as input features.

To quantify the goodness of each method in dismantling the network, we consider the *Area Under the Curve* (AUC) encoding changes in the *Largest Connected Component* (LCC) size across the attacks. The LCC size is commonly used in the literature to quantify the robustness of a network, because systems need the existence of a giant cluster to work

properly. The AUC indicator[2] has the advantage of accounting for how quickly, overall, the LCC is disintegrated: the lower the area under the curve, the more efficient is the network dismantling.

As a representative example, we show in Figure 3.2a the result of the dismantling process for the *corruption* network [218], built from 65 corruption scandals in Brazil, as a function of the number of removed units. Results are shown for GDM and for the cutting-edge algorithms mentioned above. In Figures 3.2b and 3.2c, instead, we show the structure before and after dismantling, respectively. Our framework disintegrates the network faster than other methods: to verify if this feature is general, we perform a thorough analysis of several empirical systems.



(a) Dismantling process. (b) Original network. (c) Attacked network.

Fig. 3.2 **Dismantling the Brazilian corruption network.** (a) GDM and state-of-the-art algorithms with reinsertion of the nodes are compared. The network before (b) and after (c) a GDM attack is shown. The color of the nodes represents (from dark red to white) the attack order, while their size represents their betweenness value. In the attacked network, darker nodes do not belong to the LCC, and their contour color represents the component they belong to.

3.3.1 Dismantling empirical systems

Figure 4.2 shows the performance of each dismantling method on each empirical system considered in this study, allowing for an overall comparison. On average, our approach outperforms the others. For instance, *Generalized Network Dismantling*'s cumulative AUC is $\sim 12\%$ higher and the Min-Sum algorithm is outscored by a significant margin, which is remarkable considering that our approach is static — i.e., predictions are made at the

[2]We compute the AUC value by integrating the $LCC(x)/|N|$ values using Simpson's rule.

beginning of the attack — while the other ones are dynamic — i.e., structural importance of the nodes is (re)computed during the attacks. For a more extensive comparison with these approaches, we also introduce a node reinsertion phase using a greedy algorithm which reinserts, a posteriori, those nodes that belong to smaller components of the (virtually) dismantled system and which removal is not actually needed in order to reach the desired target [204]. Once again, our approach outperforms the other algorithms: even without accounting for the reinsertion phase, GDM performs comparably with GND + reinsertion and outscores the others, highlighting how it is able to identify the more critical nodes of a network.

Heuristic Network	GDM	GND	EGND	Adaptive degree	EI σ_1	Pagerank	Degree	Betweenness	MS	EI σ_2	GDM +R	GND +R	CoreHD	MS +R	CI $\ell-2$
ARK201012_LCC	100.0	99.7	100.1	103.3	128.4	103.1	104.9	123.3	130.9	3883.7	94.5	87.6	92.6	95.8	114.6
advogato	100.0	108.0	105.5	101.8	111.6	150.1	113.4	114.8	112.6	494.5	94.8	97.5	102.1	102.7	98.8
arenas-meta	100.0	129.0	141.9	103.6	120.4	114.8	116.4	142.4	120.5	579.3	90.8	92.5	95.5	94.9	96.9
cfinder-google	100.0	160.4	246.5	99.5	233.7	113.5	141.3	377.9	682.8	1609.3	67.5	105.6	101.0	166.9	114.0
corruption	100.0	99.3	126.9	157.3	236.3	147.5	400.1	166.7	864.8	1141.9	97.6	147.4	138.6	139.6	176.6
dblp-cite	100.0	113.3	121.7	113.5	111.7	114.7	131.6	119.0	139.8	533.5	103.9	108.5	132.3	132.5	117.1
dimacs10-celegansneural	100.0	85.0	95.9	103.1	105.7	116.4	120.8	125.1	117.5	182.2	94.2	103.8	111.6	110.3	99.7
dimacs10-polblogs	100.0	107.5	97.1	102.1	115.5	112.5	117.9	114.8	107.5	262.3	98.4	108.4	106.0	104.9	104.6
econ-wm1	100.0	130.3	114.4	109.8	128.0	131.0	129.4	132.7	107.7	309.3	99.6	109.4	106.0	105.9	126.3
ego-twitter	100.0	116.8	115.8	108.9	103.0	107.8	108.8	133.3	167.3	6017.4	98.8	98.2	114.4	111.7	103.9
eu-powergrid	100.0	75.9	89.1	138.8	73.8	180.1	163.5	174.5	290.9	3313.0	64.4	66.5	83.4	92.8	109.4
foodweb-baydry	100.0	104.5	99.5	98.1	103.0	120.5	122.3	109.4	104.4	125.2	97.8	98.0	101.2	99.3	110.6
foodweb-baywet	100.0	110.2	108.4	99.6	103.9	123.6	125.4	112.9	106.8	128.3	98.5	108.5	102.1	101.8	113.0
inf-USAir97	100.0	112.4	117.8	130.4	147.0	117.1	139.1	128.6	164.0	633.6	100.1	117.2	103.7	107.6	129.8
internet-topology	100.0	95.6	95.8	99.1	113.9	109.2	131.4	122.9	138.6	3879.9	94.8	84.7	100.2	101.7	103.0
librec-ciaodvd-trust	100.0	113.1	115.5	117.6	129.4	120.5	139.8	114.9	126.6	634.5	104.3	114.4	124.4	126.3	126.1
librec-filmtrust-trust	100.0	108.9	118.3	117.7	112.8	131.8	148.4	158.9	168.7	1308.2	89.7	95.5	106.8	98.6	98.0
linux	100.0	97.9	101.1	116.2	84.5	176.0	190.8	365.1	150.0	1035.2	78.3	71.4	74.1	80.1	92.1
loc-brightkite	100.0	100.2	100.3	98.6	97.7	104.3	110.9	122.1	106.7	593.9	89.5	99.7	92.1	92.4	93.0
maayan-Stelzl	100.0	144.1	133.0	102.5	114.3	113.4	127.7	137.0	111.7	1269.6	96.3	113.4	107.1	105.2	105.4
maayan-figeys	100.0	104.3	120.2	100.7	155.9	127.3	146.9	153.4	129.5	1656.6	98.0	100.1	123.7	123.4	99.5
maayan-foodweb	100.0	111.5	94.6	114.7	147.8	118.9	123.8	126.2	154.6	268.7	100.0	125.5	136.1	144.4	173.9
maayan-vidal	100.0	111.0	106.7	103.3	101.6	109.1	110.6	123.9	114.1	843.9	90.1	102.5	95.6	97.9	97.3
moreno_crime_projected	100.0	105.8	86.0	191.2	139.2	157.6	218.8	180.6	976.7	2103.3	82.7	88.8	100.3	104.1	126.2
moreno_propro	100.0	115.9	123.6	115.6	87.9	126.1	123.5	146.7	145.2	1985.3	90.7	94.6	92.2	93.1	96.3
moreno_train	100.0	104.9	104.9	107.1	124.0	149.5	156.0	134.7	176.9	408.8	100.0	109.7	115.6	120.3	211.6
munmun_digg_reply_LCC	100.0	116.3	108.6	98.5	109.4	106.5	108.3	117.5	98.9	556.8	95.6	104.0	99.0	98.4	98.5
opsahl-openflights	100.0	101.2	106.2	127.2	109.9	123.2	135.4	123.6	157.3	807.7	84.4	92.0	102.6	111.3	120.9
opsahl-powergrid	100.0	36.9	69.4	148.6	37.0	173.4	180.9	183.9	164.3	1508.1	43.1	42.1	51.4	52.5	65.6
opsahl-ucsocial	100.0	122.1	116.1	99.9	118.5	105.9	109.9	109.8	108.8	342.0	97.0	106.1	105.8	106.0	101.7
oregon2_010526	100.0	106.8	101.5	108.8	131.1	101.6	130.5	114.6	162.0	3247.5	90.0	80.5	113.0	112.8	95.1
p2p-Gnutella06	100.0	128.5	120.4	108.5	108.6	111.6	125.1	118.4	108.7	274.0	101.4	120.4	110.1	108.4	109.1
p2p-Gnutella31	100.0	133.6	NaN	109.1	112.7	110.3	123.1	129.5	109.2	474.4	102.3	121.6	110.4	108.8	109.8
pajek-erdos	100.0	112.2	107.5	103.3	119.9	103.3	104.6	106.7	122.8	2790.7	98.2	106.9	116.7	113.9	101.0
petster-hamster	100.0	92.5	90.9	122.7	103.8	135.1	127.2	123.8	166.7	402.6	91.5	93.3	96.2	96.5	98.6
power-eris1176	100.0	199.1	218.0	340.2	171.7	253.5	622.5	430.2	632.6	1957.4	86.6	154.8	161.3	157.8	153.7
route-views	100.0	99.3	99.1	101.8	133.2	103.5	103.5	112.3	131.5	4340.9	94.0	82.0	93.0	95.2	112.5
slashdot-threads	100.0	100.1	102.2	99.5	122.5	104.6	105.4	114.2	117.6	1495.8	96.1	95.8	115.7	115.1	97.9
slashdot-zoo	100.0	99.2	100.1	95.6	120.9	103.3	106.8	124.0	112.4	683.8	95.2	97.7	106.9	105.9	96.5
subelj_jdk	100.0	107.6	110.2	115.1	113.0	144.2	181.5	346.9	144.6	1275.7	80.9	84.7	84.8	81.0	103.4
subelj_jung-j	100.0	102.1	111.6	122.0	118.4	150.7	185.7	334.6	143.1	1295.0	80.1	88.5	82.9	72.2	101.5
web-EPA	100.0	148.4	157.6	102.2	141.1	104.9	109.7	137.7	158.1	1471.9	101.1	115.6	133.8	132.8	107.3
web-webbase-2001	100.0	127.6	130.0	165.0	196.6	216.3	165.4	207.7	3603.1	55066.4	64.7	50.1	76.6	82.6	80.9
wikipedia_link_kn	100.0	107.3	102.6	103.7	113.2	124.8	143.6	140.3	128.8	NaN	92.9	98.0	113.9	113.5	96.8
wikipedia_link_li	100.0	120.3	145.4	132.8	151.8	120.5	165.2	110.6	211.9	1049.4	107.2	151.0	177.5	174.5	157.8
Average	100.0	111.7	115.4	119.1	123.6	128.7	151.1	158.9	273.3	2596.4	91.5	100.8	106.9	108.7	112.1

Table 3.1 Per-method area under the curve (AUC) of real-world networks dismantling. The lower the better. The dismantling target for each method is 10% of the network size. We compute the AUC value by integrating the $LCC(x)/|N|$ values using Simpson's rule, and each value is scaled to the one of our approach (GDM) for the same network. +R means that the reinsertion phase is performed. CoreHD and CI are compared to other +R algorithms as they include the reinsertion phase. EGND for p2p-Gnutella31 is missing as the computation was killed after 10d.

In Table 3.1 we report the same results in numerical form. The table also includes other commonly used static attack approaches that remove the nodes in descending importance order according to some node centrality metric. While many heuristics fall in this category, we compare with the removal of nodes in descending degree [219], betweenness [219] and PageRank [57]. Our approach outperforms all these static approaches with a significant margin, even the ones with higher computational complexity (e.g., the betweenness-based one).

For the full dismantling curves (i.e., LCC as a function of the removed nodes), we refer the Reader to the Figures 3.7 and 3.8 of Section 3.3.5.

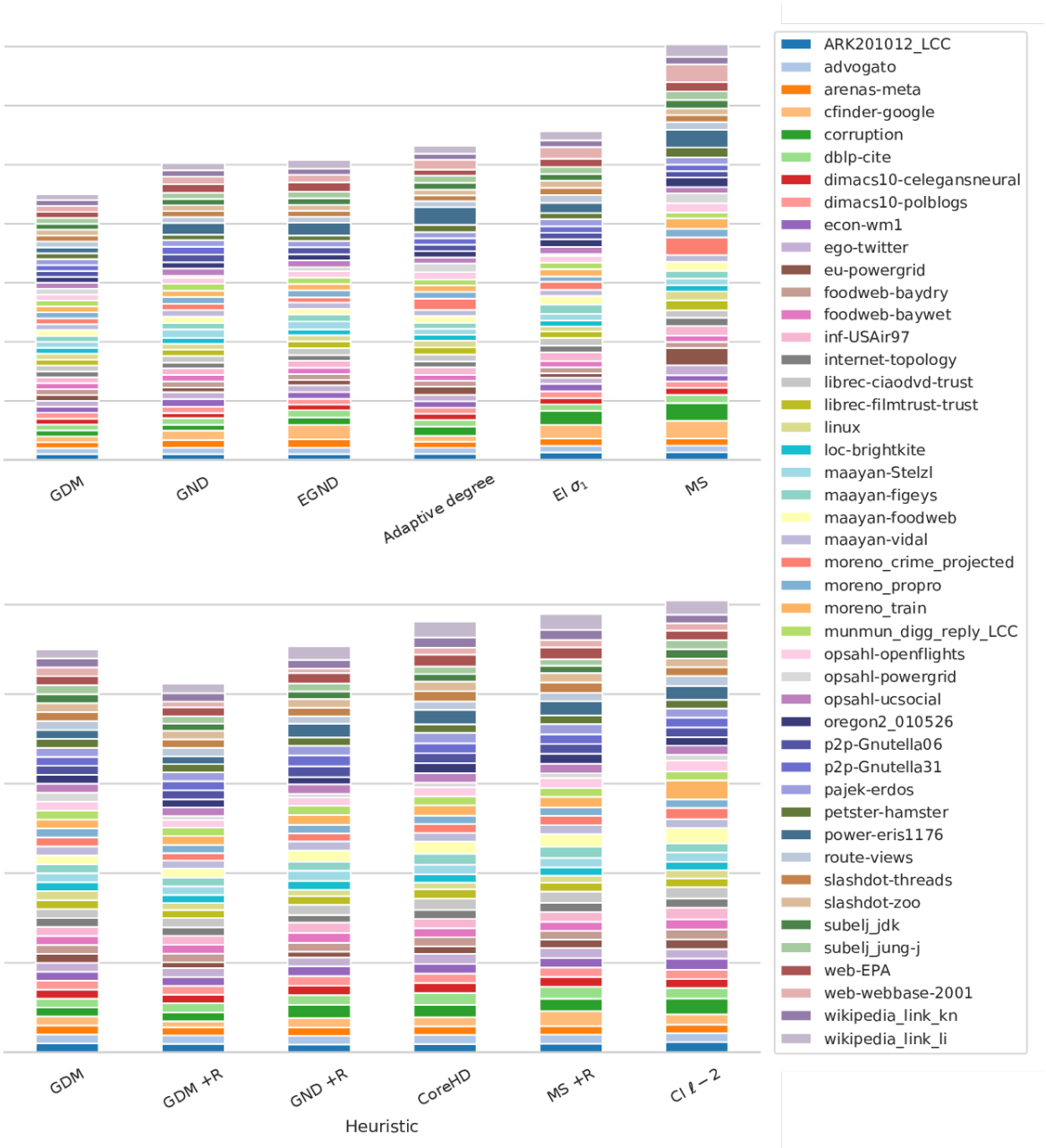


Fig. 3.3 **Dismantling empirical complex systems.** Per-method cumulative area under the curve (AUC) of real-world networks dismantling. The lower the better. The dismantling target for each method is 10% of the network size. Each value is scaled to the one of our approach (GDM) for the same network. *GND* stands for *Generalized Network Dismantling*, *EGND* for *Ensemble approach for GND* (in both *GND* and *EGND*, cost matrix $\mathbf{W} = \mathbf{I}$), *MS* stands for *Min-Sum*, *EI* σ_1 stands for *Explosive Immunization* (σ_1) algorithm and *CI* for *Collective Influence*. +R means that the reinsertion phase is performed. *CoreHD* and *CI* are compared to other +R algorithms as they include the reinsertion phase. Also, note that some values are clipped (limited) to 3x for the *MS* heuristic to improve visualization.

3.3.2 Dismantling large empirical systems

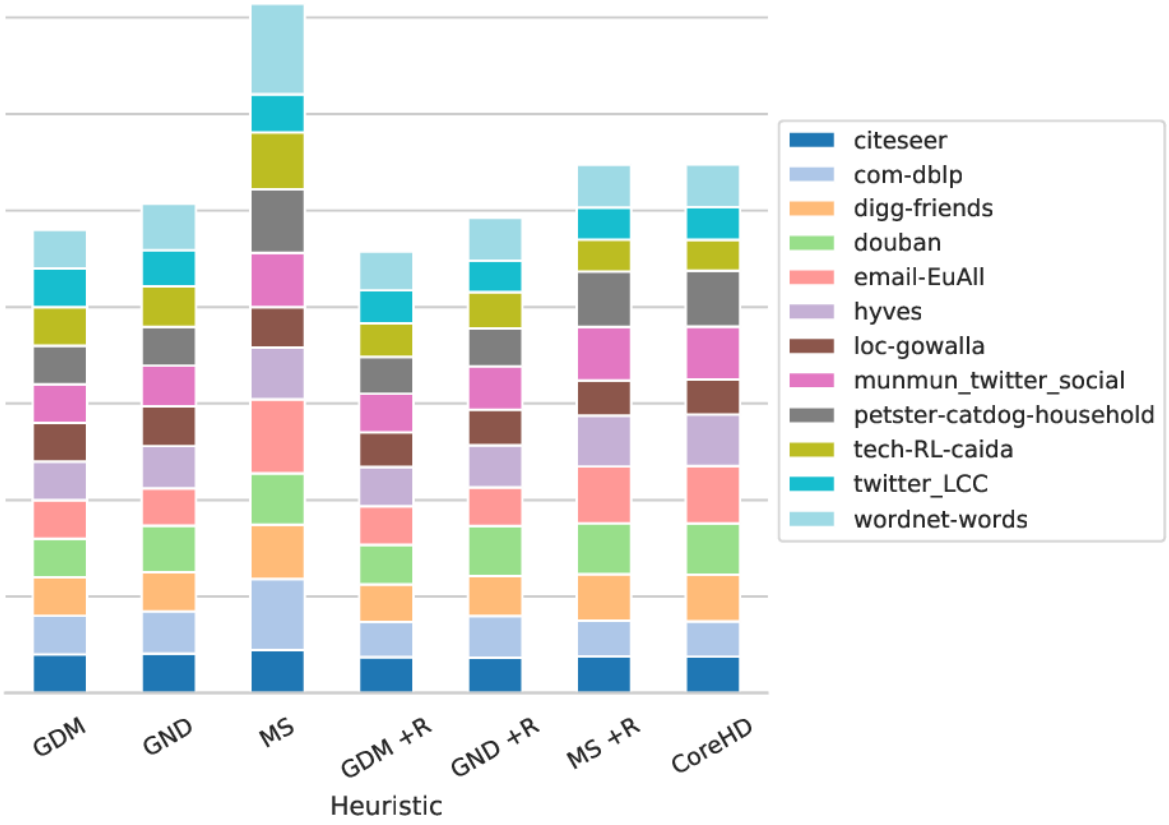


Fig. 3.4 **Dismantling empirical complex large systems.** Per-method cumulative area under the curve (AUC) of real-world networks dismantling. The lower the better. The dismantling target for each method is 10% of the network size. We compute the AUC value by integrating the $LCC(x)/|N|$ values using Simpson's rule, and each value is scaled to the one of our approach (GDM) for the same network. *GND* stands for *Generalized Network Dismantling* (with cost matrix $\mathbf{W} = \mathbf{I}$) and *MS* stands for *Min-Sum*. +R means that the reinsertion phase is performed. Also, note that some values are clipped (limited) to 3x for the *MS* heuristic to improve visualization.

We extend the comparison against the more promising state-of-the-art algorithms (*GND* and *MS* with and without reinsertion, and *CoreHD*) to 12 large networks with up to 1.8M nodes and up to 2.8M edges. As shown in Figure 3.4, the results on smaller empirical networks are confirmed even for the large ones, although with smaller margins (i.e., $\sim 5.6\%$ and $\sim 5.6\%$ against GND, respectively with and without the reinsertion phases) This is still impressive as the proposed approach is static while the others recompute the nodes' structural importance during the dismantling process, which involves many removals for these networks (e.g., 70K on *hyves* network) and changes the network topology drastically, confirming the validity of our approach.

Heuristic Network	GDM	GND	MS	GDM +R	GND +R	MS +R	CoreHD
citeseer	100.0	102.2	111.2	92.8	91.3	95.0	94.3
com-dblp	100.0	109.6	184.5	91.5	108.3	92.4	91.2
digg-friends	100.0	100.9	140.5	97.0	103.6	120.7	121.2
douban	100.0	120.8	132.7	102.6	129.3	131.6	132.9
email-EuAll	100.0	97.0	192.1	100.0	100.0	147.4	148.5
hyves	100.0	109.3	133.6	101.6	109.6	131.9	133.6
loc-gowalla	100.0	103.2	105.4	89.7	91.9	91.0	90.5
munmun_twitter_social	100.0	105.2	140.5	100.2	112.4	138.5	137.3
petster-catdog-household	100.0	100.7	164.7	95.4	98.0	143.4	144.7
tech-RL-caida	100.0	104.8	147.2	86.9	94.3	82.8	80.2
twitter_LCC	100.0	93.6	98.8	85.3	81.4	83.0	84.7
wordnet-words	100.0	120.4	234.5	100.0	110.8	111.0	109.7
Average	100.0	105.6	148.8	95.3	102.6	114.1	114.1

Table 3.2 Per-method area under the curve (AUC) of real-world large networks dismantling. The lower the better. The dismantling target for each method is 10% of the network size. We compute the AUC value by integrating the $LCC(x)/|N|$ values using Simpson's rule, and each value is scaled to the one of our approach (GDM) for the same network. +R means that the reinsertion phase is performed. CoreHD and CI are compared to other +R algorithms as they include the reinsertion phase.

In Table 3.3, we also report the prediction (if any) and dismantling time of each of the above-mentioned methods to give a better idea on what their different computational complexities mean and translate into.

	Prediction time		Dismantle time		
Heuristic Network	GDM	CoreHD	GDM	GND	MS
citeseer	00:00:03.4	00:00:22.9	01:30:17.1	03:43:51.6	01:26:21.5
com-dblp	00:00:02.9	00:00:14.9	00:22:30.7	04:57:25.6	00:59:38.4
digg-friends	00:00:02.8	00:00:19.9	00:08:01.9	00:30:55.5	01:11:37.4
douban	00:00:01.3	00:00:06.1	00:01:10.1	00:03:34.8	00:11:40.4
email-EuAll	00:00:02.4	00:00:07.8	00:00:10.7	00:01:14.9	00:09:49.0
hyves	00:00:13.5	00:00:36.6	03:08:02.7	08:21:22.8	02:03:26.9
loc-gowalla	00:00:02.0	00:00:15.9	00:17:22.3	01:27:28.0	00:46:15.0
munmun_twitter_social	00:00:04.3	00:00:14.3	00:00:53.5	00:07:53.4	00:29:13.9
petster-catdog-household	00:00:03.9	00:00:40.6	00:44:20.5	03:58:17.1	02:16:02.8
tech-RL-caida	00:00:01.8	00:00:12.1	00:07:23.7	04:14:34.1	00:29:30.8
twitter_LCC	00:00:04.4	00:00:13.0	00:32:01.0	05:33:36.3	00:19:18.8
wordnet-words	00:00:01.4	00:00:12.1	00:03:34.0	01:23:52.1	00:22:28.5

Table 3.3 Real-world large networks dismantling timings. The lower the better. Time format is HH:MM:SS.s. *MS* and *GND* do not have prediction time as they refresh the predictions during the dismantling, while there is no *CoreHD* dismantling column as we use our dismantler.

3.3.3 Dismantling synthetic systems

We also validate the approach on synthetic networks. Specifically, we test on Erdős-Rényi (ER) networks (average degree $k_{avg} = 4$), on Configuration Model networks (CM) with power law distribution ($\gamma = 2.5$ and $k_{avg} = 4$) and on Stochastic Block Model (SBM) networks (group size fixed to 100, $p_{intra} = 0.1$ and $p_{inter} = \frac{5}{|N|}$). We generate 10 realizations with $1K$, $10K$ and $100K$ nodes each and average the results.

As reported in Figure 3.5 and Table 3.4, this time the best approach is *Min-Sum*, scoring 6% and 3% lower AUC than *GDM* and *GDM+R*, respectively.

The reason behind this slightly lower *GDM* performance can be found in our training set and on what the models learn. Specifically, we train on networks generated using three different models, which teaches the models to look for patterns that turn out to be sub-optimal in the long term (as no re-computation is made during the process) when it comes to specific synthetic networks. It should also be noted that *GND* — the second best-performing algorithm on real-world networks — is the worst of the tested algorithms on synthetic networks.

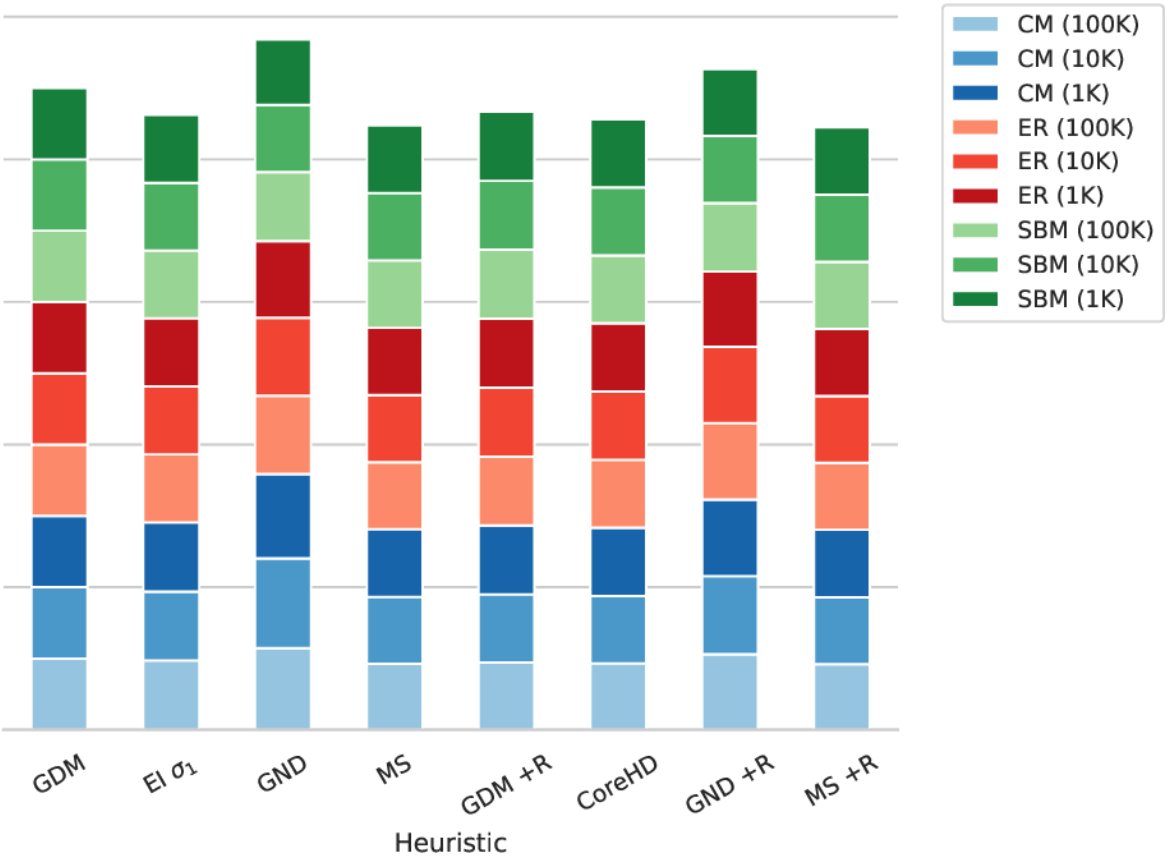


Fig. 3.5 **Dismantling synthetic complex systems.** Per method cumulative area under the curve (AUC) of the dismantling of synthetic networks. The lower the better. Each value is the average on 10 different instances, and is scaled to the AUC of our approach (GDM) for the same network type. CM stands for Configuration Model, ER stands for Erdős-Rényi, and SBM stands for Stochastic Block Model.

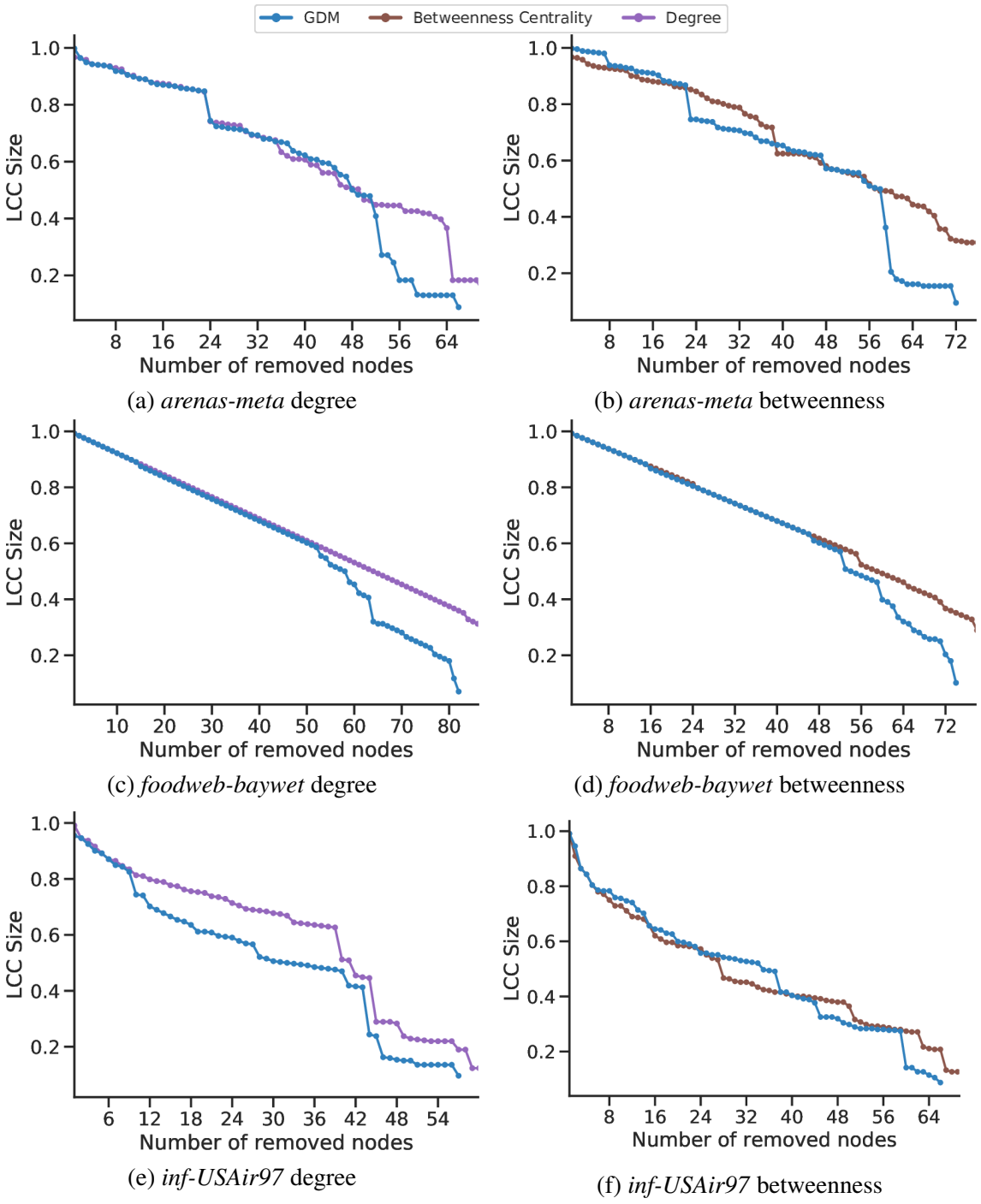
Heuristic Network	GDM	EI σ_1	GND	MS	GDM +R	CoreHD	GND +R	MS +R
CM (100K)	100.0	97.3	114.5	92.6	94.5	93.2	106.0	92.3
CM (10K)	100.0	96.7	126.1	93.9	95.5	94.5	109.5	93.6
CM (1K)	100.0	96.9	118.2	95.1	96.6	95.6	107.5	94.8
ER (100K)	100.0	95.5	109.7	93.9	96.6	95.4	107.2	93.8
ER (10K)	100.0	95.3	109.5	93.9	96.6	95.6	106.9	93.7
ER (1K)	100.0	95.5	107.3	94.5	96.6	95.8	105.4	94.2
SBM (100K)	100.0	95.0	96.9	94.5	96.7	95.2	96.0	94.1
SBM (10K)	100.0	94.8	94.0	94.5	96.7	95.3	94.4	94.1
SBM (1K)	100.0	95.2	91.9	94.9	96.9	95.4	93.5	94.4
Average	100.0	95.8	107.6	94.2	96.3	95.1	102.9	93.9

Table 3.4 **Synthetic network results table.** Per method area under the curve (AUC) of the dismantling of synthetic networks. The lower the better. Each value is the average on 10 different instances, which is scaled to the AUC of our approach (GDM) for the same network type.

3.3.4 Enhancement of node metric based heuristics

An interesting feature of our framework is that it can enhance existing heuristics based on node descriptors, by employing the same measure as the only node feature, as shown in Fig. 3.6.

Specifically, in order to better understand how our framework is able to outperform cutting-edge algorithms, we compare existing node metric-based heuristics (e.g., removal of nodes in degree order) against GDM models that employ the corresponding node metric as the only node feature. As an example, in Figure 3.6 we display the enhancement of the degree and the betweenness based heuristics in the left and right columns respectively. These GDM-enhanced heuristics effectively outperform the vanilla ones, highlighting the fact that the model is able to capture the importance of the nodes thanks to the feature propagation discussed before. This also gives an important insight as the model seems to learn correlations between node features.



(a) *arenas-meta* degree

(b) *arenas-meta* betweenness

(c) *foodweb-baywet* degree

(d) *foodweb-baywet* betweenness

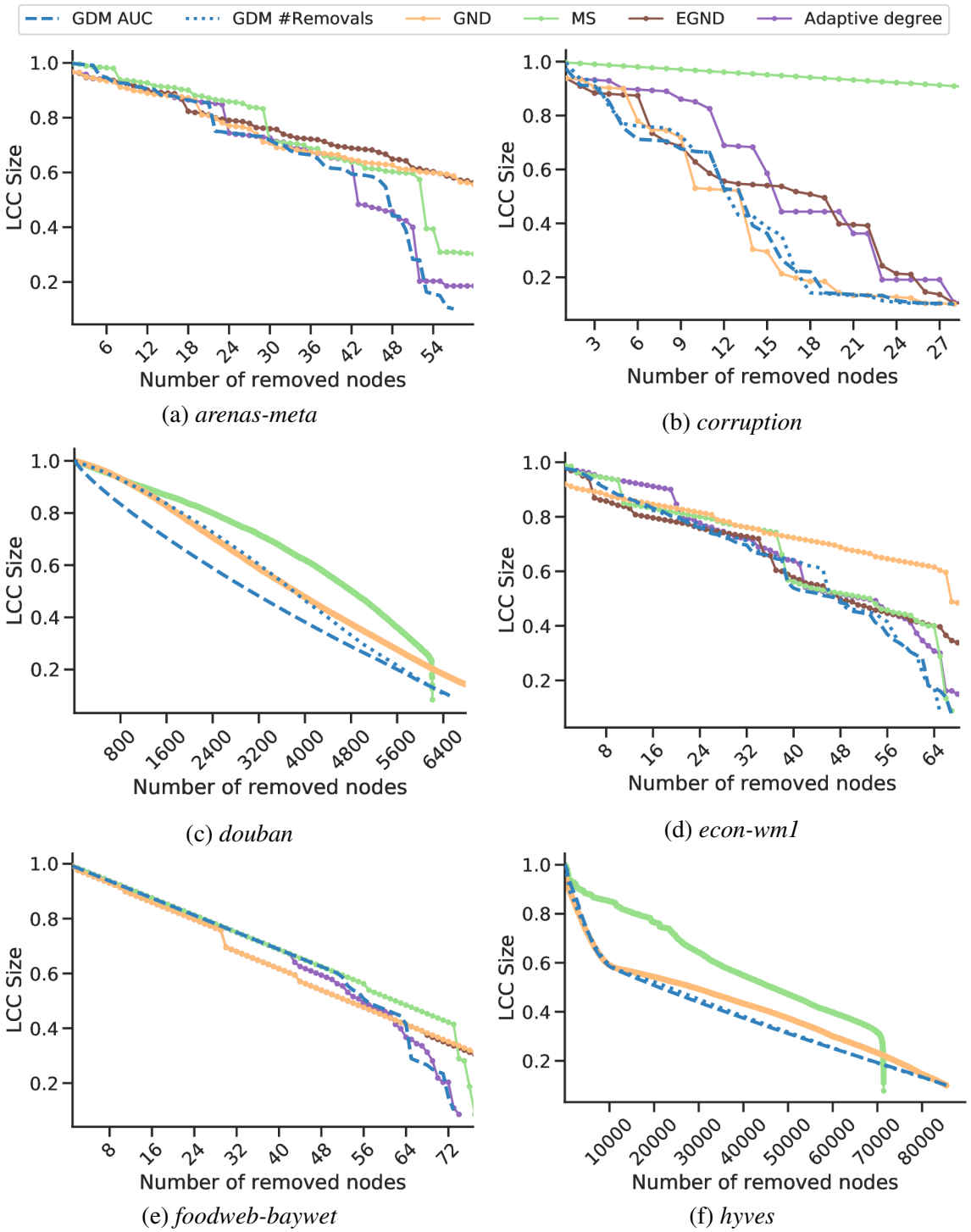
(e) *inf-USAir97* degree

(f) *inf-USAir97* betweenness

Fig. 3.6 **Heuristics enhancement.** Comparison of degree and betweenness vanilla heuristics with their GDM-enhanced versions on the *arenas-meta*, *foodweb-baywet* and *inf-USAir97* networks.

3.3.5 Dismantling curves

In Figure 3.7, we display the dismantling of most of our test networks and compare with the state-of-the-art algorithms and with the heuristics introduced in the previous paragraph. As previously mentioned, one of the advantages of our approach is that we can choose the best model to reach a given objective. As an example, we show the models that lower the area under the curve (GDM AUC) and the removals number (GDM #Removals), which may overlap for some networks. We also show the dismantling performing the reinsertion phase and compare with state-of-the-art algorithms in Figure 3.8.



(a) *arenas-meta*

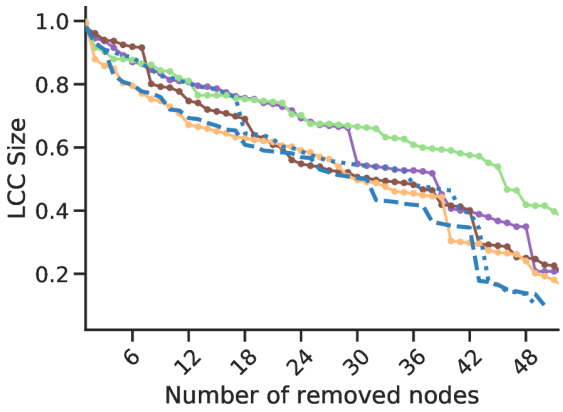
(b) *corruption*

(c) *douban*

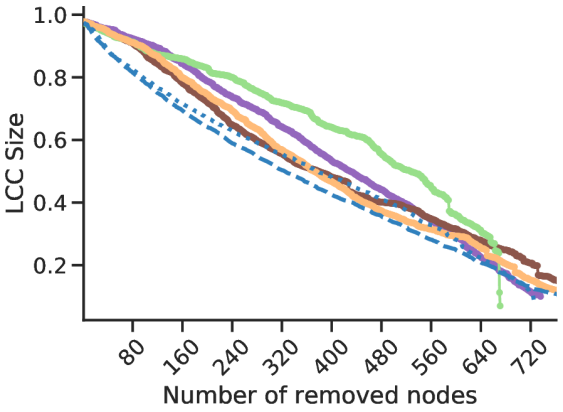
(d) *econ-wm1*

(e) *foodweb-baywet*

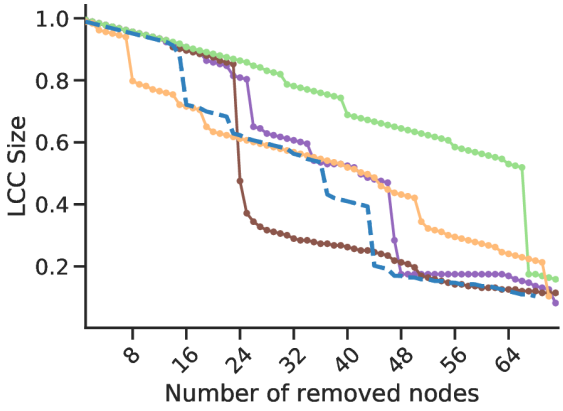
(f) *hyves*



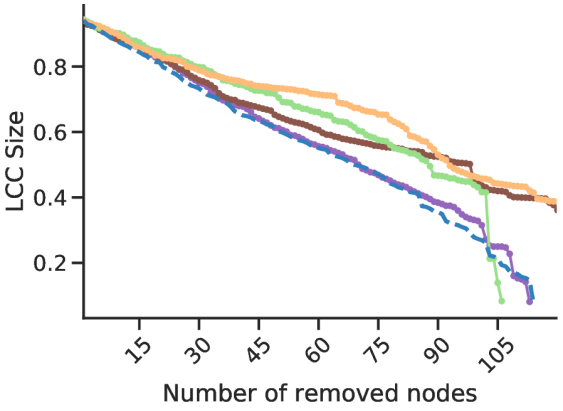
(g) *inf-USAir97*



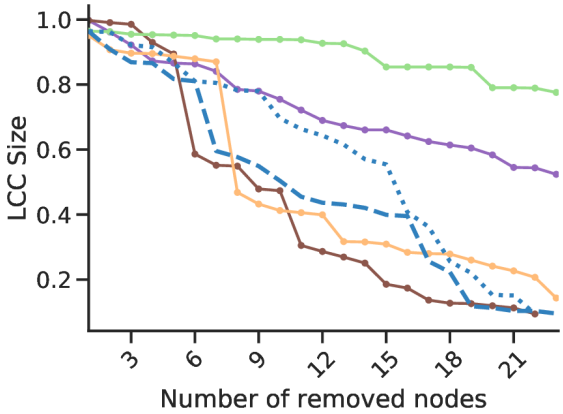
(h) *librec-ciaodvd-trust*



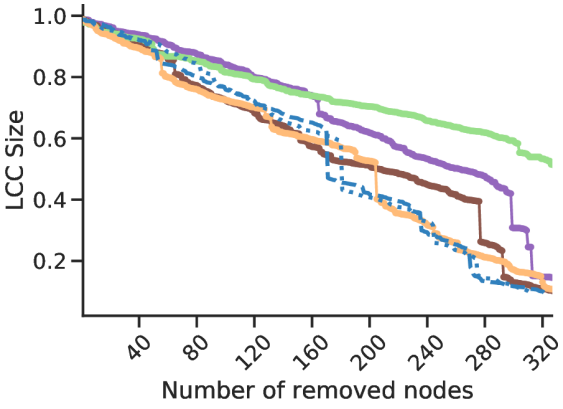
(i) *maayan-foodweb*



(j) *maayan-Stelzl*



(k) *moreno-crime-projected*



(l) *opsahl-openflights*

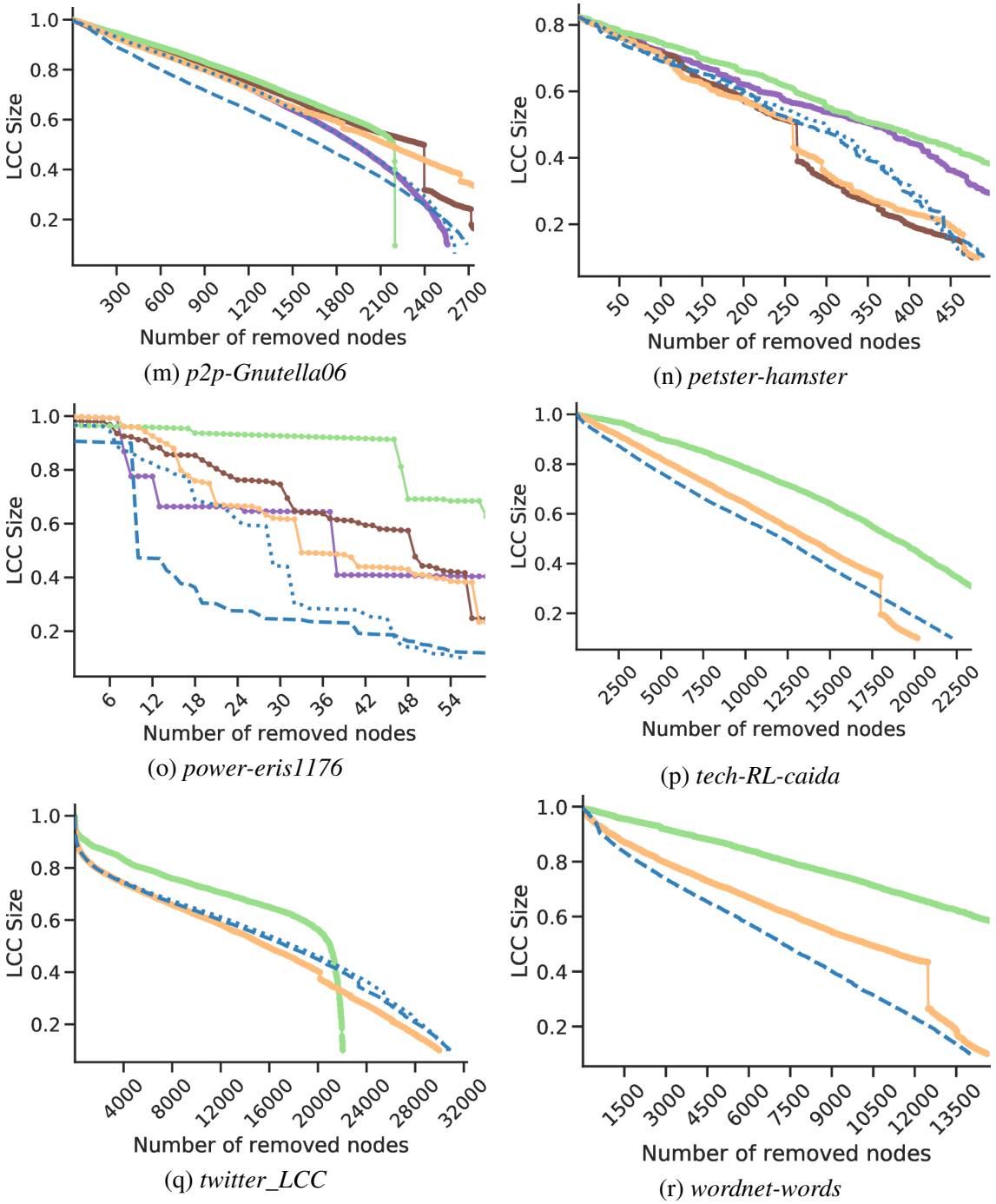
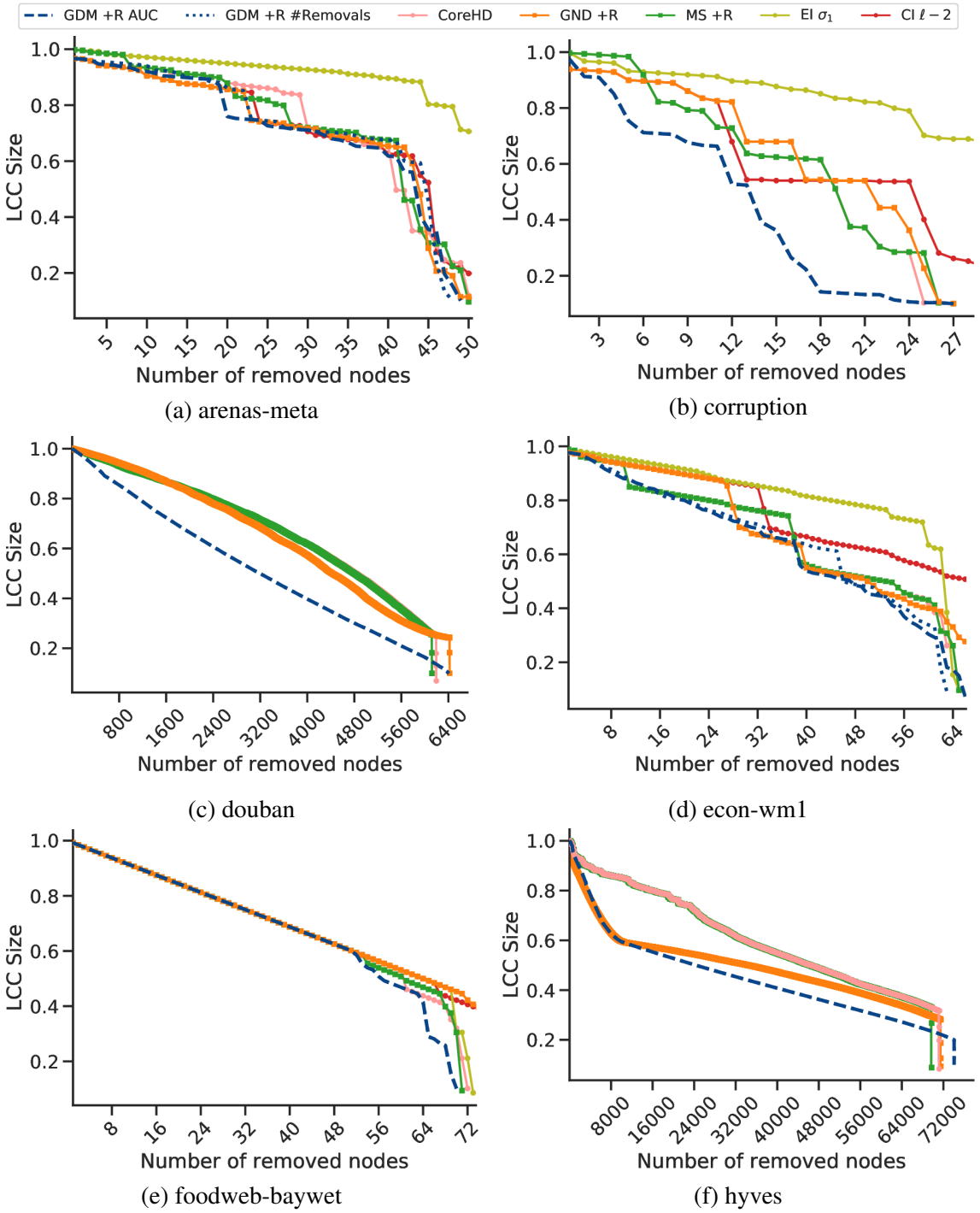


Fig. 3.7 **Dismantling curves without reinsertion phase.** Dismantling of some networks in our test set. We compare against the algorithms without reinsertion in Tables 3.1 and 3.2 and show both the models with lower area under the curve (GDM AUC) and with lower number of removals (GDM #Removals), which may overlap for some networks.



(a) arenas-meta

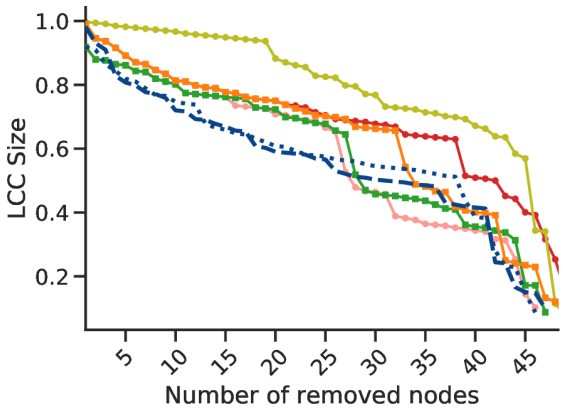
(b) corruption

(c) douban

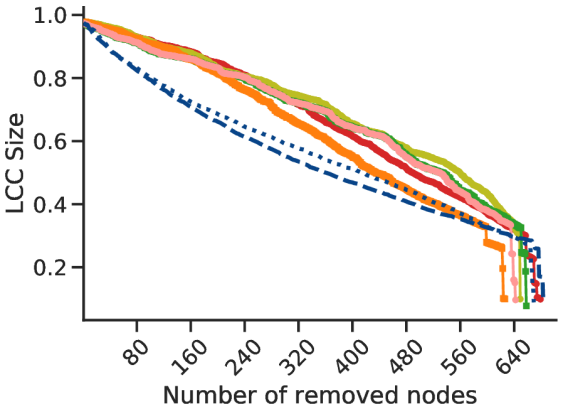
(d) econ-wm1

(e) foodweb-baywet

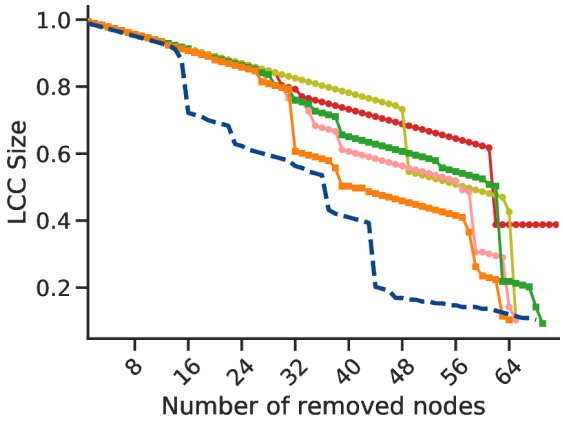
(f) hyves



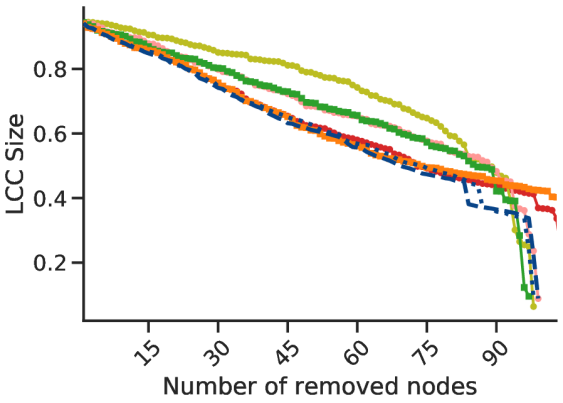
(g) inf-USAir97



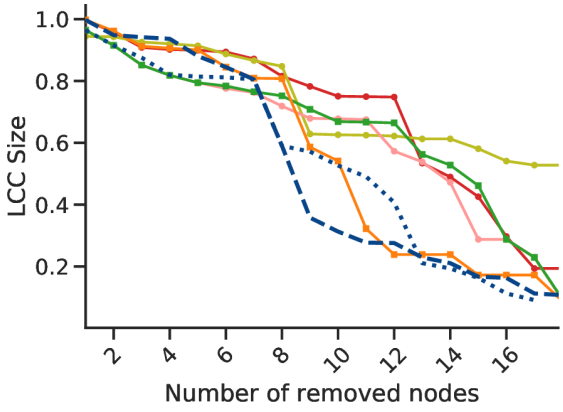
(h) librec-ciaodvd-trust



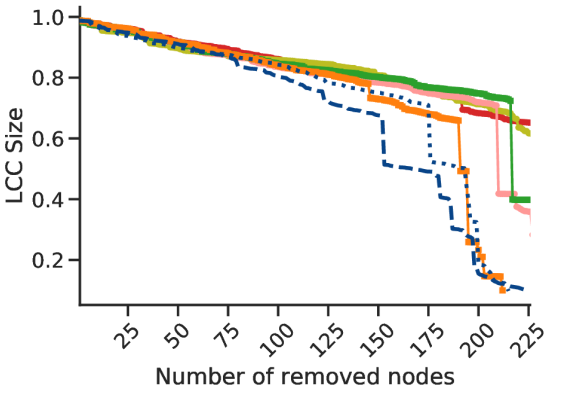
(i) maayan-foodweb



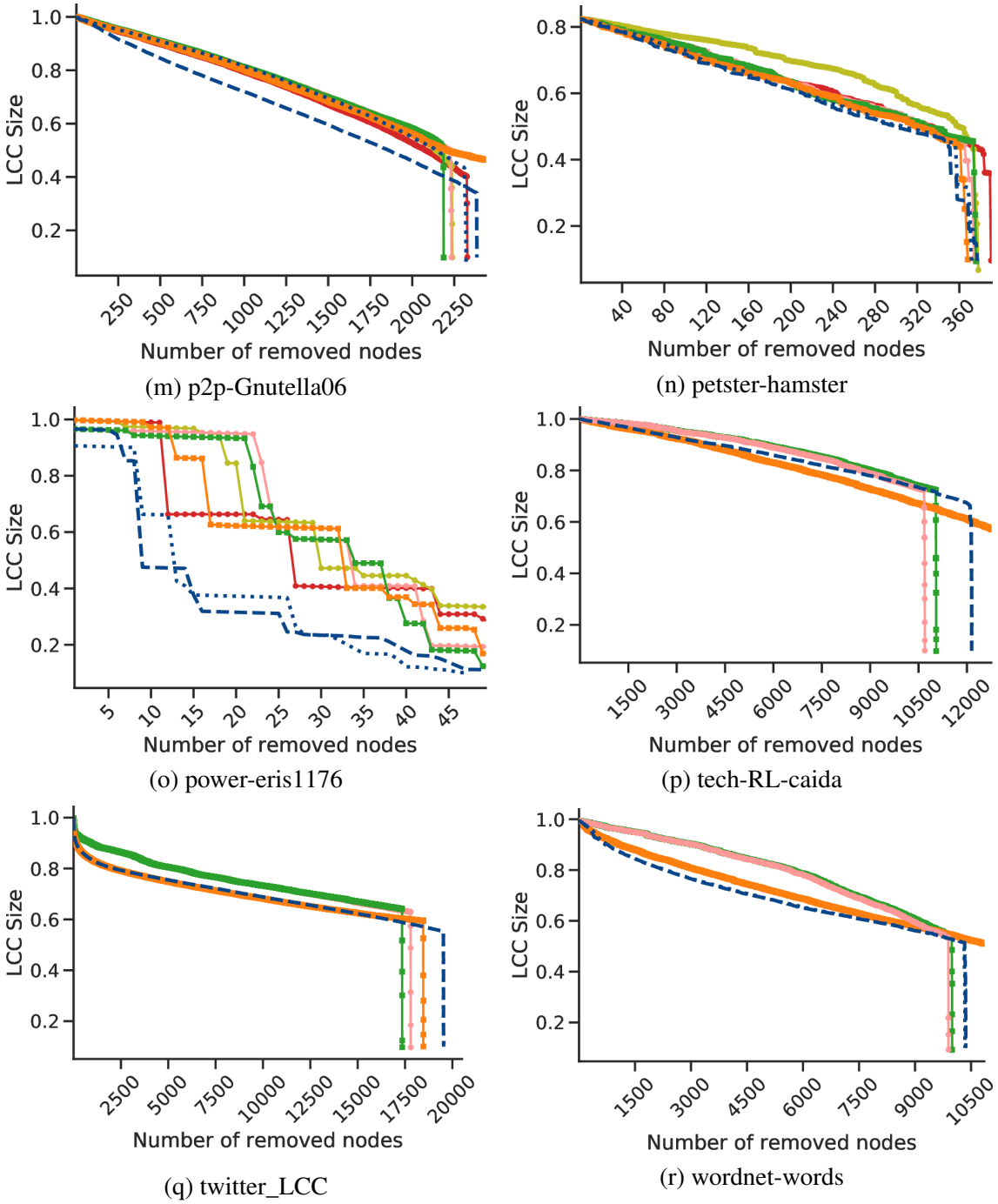
(j) maayan-Stelzl



(k) moreno-crime-projected



(l) opsahl-openflights



(m) p2p-Gnutella06 (n) petster-hamster

(o) power-eris1176 (p) tech-RL-caida

(q) twitter_LCC (r) wordnet-words

Fig. 3.8 **Dismantling curves with reinsertion phase.** Dismantling of some networks in our test set. We compare against the algorithms with reinsertion phase in Tables 3.1 and 3.2 and show both the models with lower area under the curve (GDM +R AUC) and with lower number of removals (GDM +R #Removals), which may overlap for some networks.

3.4 Early-warning signals of systemic collapse

Another relevant output of our method is the calculation of a damage score that can be used to predict the impact of future attacks to the system. Accordingly, we introduce an estimator of early warning that can be used for inform policy and decision making in applications where complex interconnected systems – such as water management systems, power grids, communication systems and public transportation networks – are subject to potential failures or targeted attacks. We define Ω, namely *Early Warning*, as a value between 0 and 1, calculated as follows. We first simulate the dismantling of the target network using our approach and call S_o the set of virtually removed nodes that cause the percolation of the network. Then, we sum the p_n values predicted by our model for each node $n \in S_o$ and define

$$\Omega_m = \sum_{n \in S_o} p_n. \tag{3.1}$$

The value of the Early Warning Ω for the network after the removal of a generic set S of nodes is given by

$$\Omega = \begin{cases} \Omega_s / \Omega_m & \text{if } \Omega_s \leq \Omega_m \\ 1 & \text{otherwise} \end{cases} \tag{3.2}$$

where $\Omega_s = \sum_{n \in S} p_n$.

The rationale behind this definition is that the system will tolerate a certain amount of damage before it collapses: this value is captured by Ω_m. Ω will quickly reach values close to 1 when nodes with key-role in the integrity of the system are removed. Of course, the system could be heavily harmed by removing many less relevant nodes (e.g., the peripheral ones) with an attack that causes a small decrease in LCC size over time, and probably get a low value of Ω. However, this kind of attacks does not need an early-warning signal since they do not cause an abrupt disruption of the system and can be easily detected.

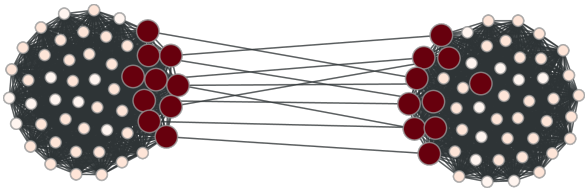
3.4.1 Why do we need an Early Warning signal?

One may wonder why do we need an Early Warning signal, since it is common to evaluate the structural integrity using, for instance, the Largest Connected Component of the system.

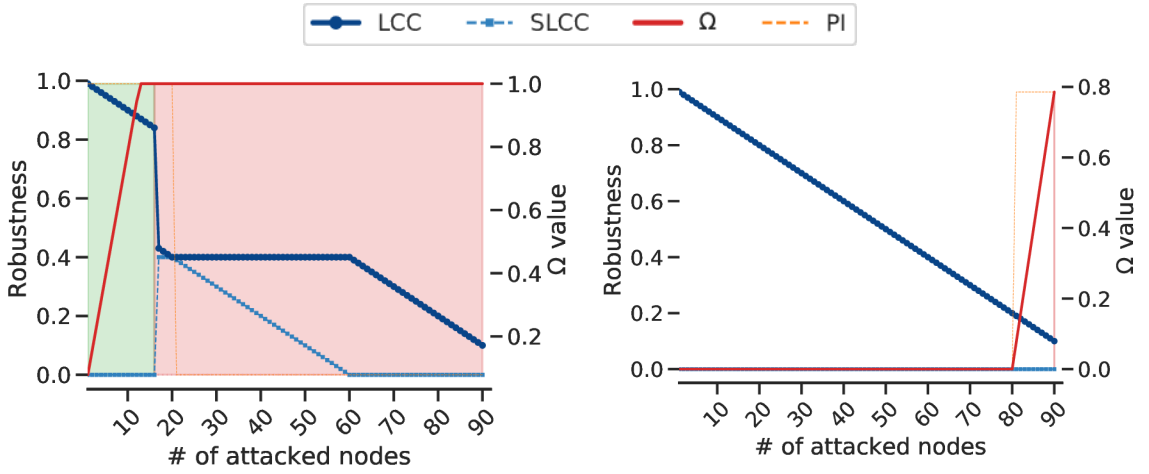
In this Section we try to answer with a simple toy-example shown in Figure 3.9.

Specifically, the toy-example network in Figure 3.9a is composed of two cliques (fully connected sub-networks) connected by a few border nodes (bridges) that also belong to the respective cliques. Many dismantling approaches (like the degree and betweenness-based heuristics, or even ours) would remove those bridge nodes first, meaning that the network

would eventually break in two, as shown in Figure 3.9b. Now, when most of the bridge nodes are removed (e.g., after 16 removals), the LCC is still quite large as it includes more than 80% of the nodes, but it takes just a few more removals of the bridges to break the network in two. While Ω is able to capture the imminent system disruption (i.e., the Ω value gets closer to 1 very fast), the LCC size is not, and one would notice when it is too late. Moreover, the LCC curve during the initial part of the attack is exactly the same as the one in Figure 3.9c, showing the removal of nodes in inverse degree (or betweenness) order, which does not cause the percolation of the system. Again, Ω captures this difference and does not grow, meaning that a slow degradation should be expected.



(a) Toy-example network composed of two cliques connected by 10 bridges. The size of the nodes represents their betweenness value and the color (from dark red to white) represents their importance to the system's health according to our method.



(b) Degree or betweenness based attack.

(c) Inverse degree or betweenness based attack.

Fig. 3.9 **Toy-example meant to explain why the LCC is not sufficient to evaluate the state of the system.** The LCC decreases at the same rate during the initial part of both the attacks shown. Instead, Ω values do not and reach warning levels before the system suddenly collapses.

3.4.2 Tests on real-world systems

We test our method on key infrastructure networks and predict the collapse of the system under various attack strategies (see Fig. 3.10 for details). Remarkably, while the LCC size decreases slowly without providing a clear alarm signal until the system is heavily damaged and collapses, Ω grows faster when critical nodes are successfully attacked, reaching warning levels way before the system is disrupted, as highlighted by the First Response Time, defined as the time occurring between system's collapse and an early-warning signal of 50% (i.e., $\Omega = 0.5$). Moreover, the first order derivative Ω'_s tracks the importance of nodes that are being attacked, providing a measure of the attack intensity over time.

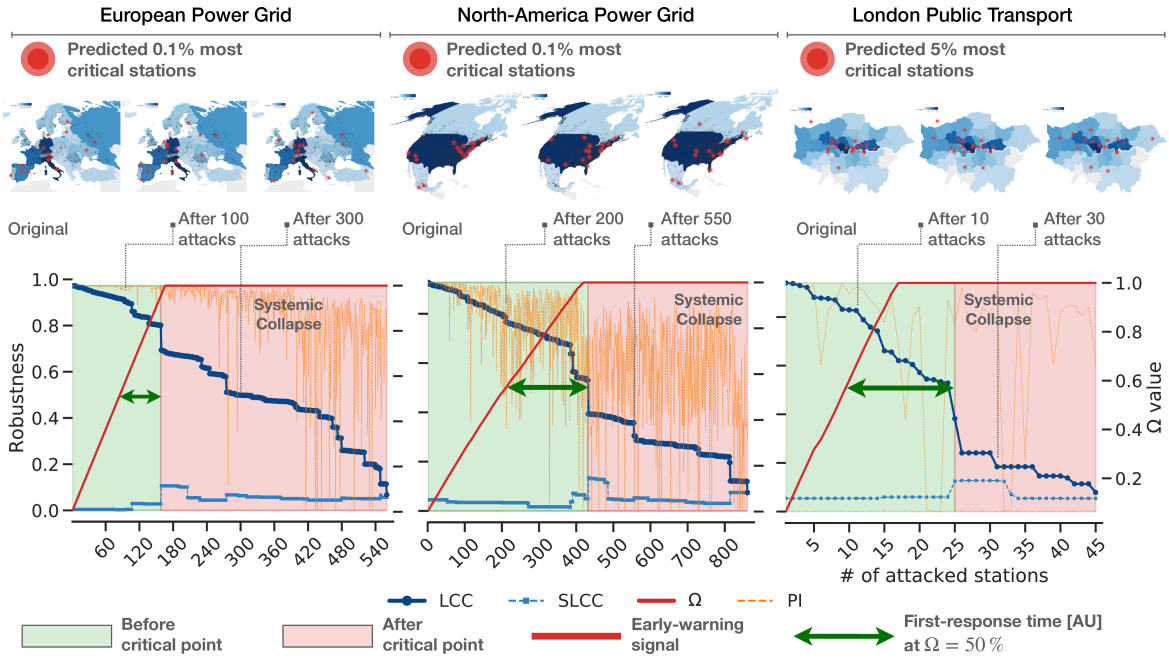
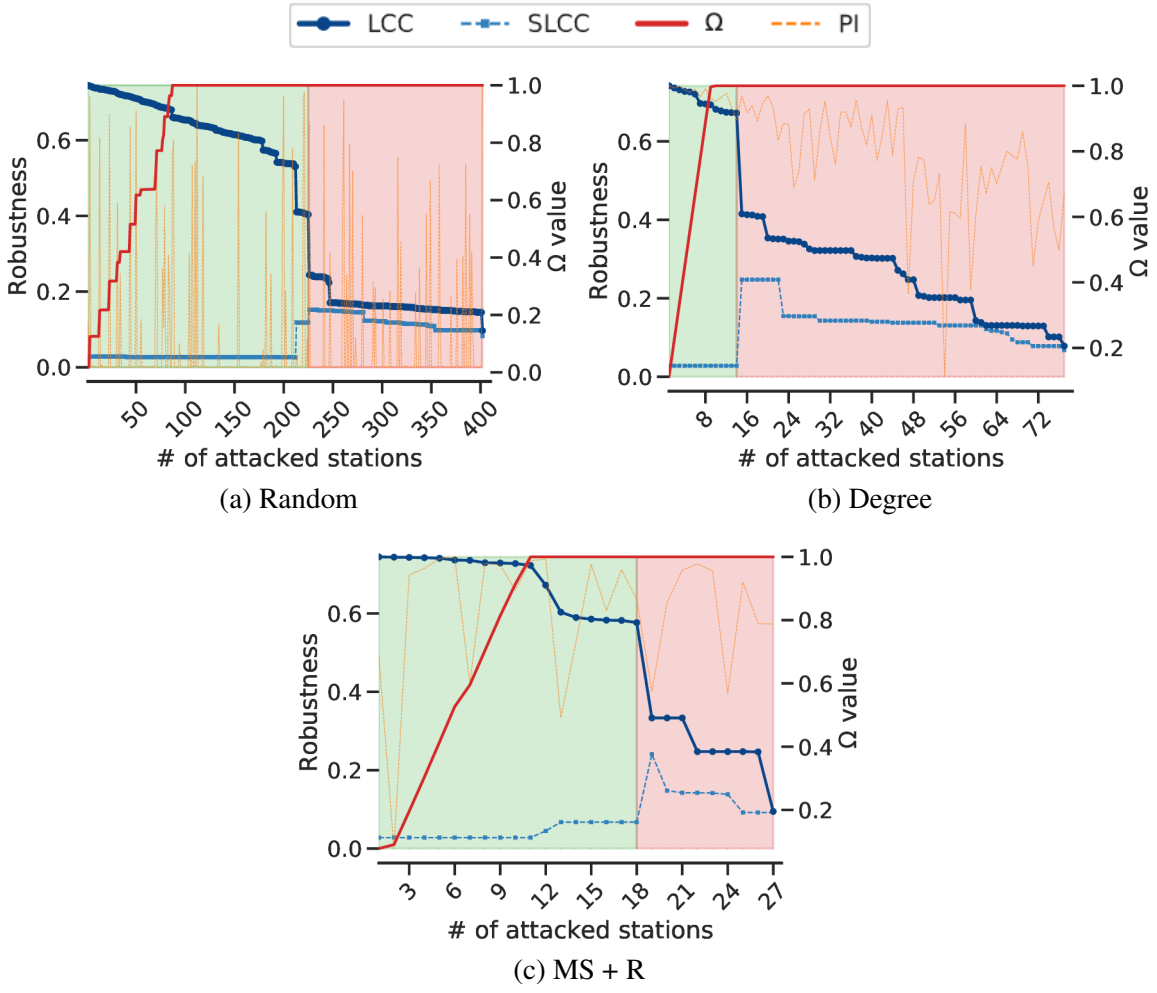


Fig. 3.10 **Early warning due to network dismantling of real infrastructures.** Three empirical systems, namely the European power grid (left), the North-American power grid (middle) and the London public transport (right), are repeatedly attacked using a degree-based heuristics, i.e., hubs are damaged first. A fraction of the most vulnerable stations is shown for the original systems and some representative damaged states (i.e., before and after the critical point for system's collapse), in the top of the figure. The plots show the behavior of the largest (LCC) and second-largest (SLCC) connected components, as well as the behavior of Ω, the Early Warning descriptor introduced in this study and the p_n value of each removed node (PI). Transitions between green and red areas indicate the percolation point of the corresponding systems, found through the SLCC peak. We also show the first response time in arbitrary units (AU), to highlight how our framework allows to anticipate system's collapse, allowing for timely emergency response.

3.4.3 More Early Warning Ω examples

In addition to the example applications of Ω illustrated above, we also test if it can detect the collapse of other systems. Specifically, we show the SciKit European power-grid (*eu-powergrid*) under random failures, degree or Min-Sum + Reinsertion phase attacks in Figure 3.11, and also various American roads under Generalized Network Dismantling + Reinsertion phase attacks in Figure 3.12. In all these scenarios, Ω is able to detect the system damage and reaches warning levels before the system collapse actually happens, even in case of multiple large connected components detaching from the larger one as the attack goes on.

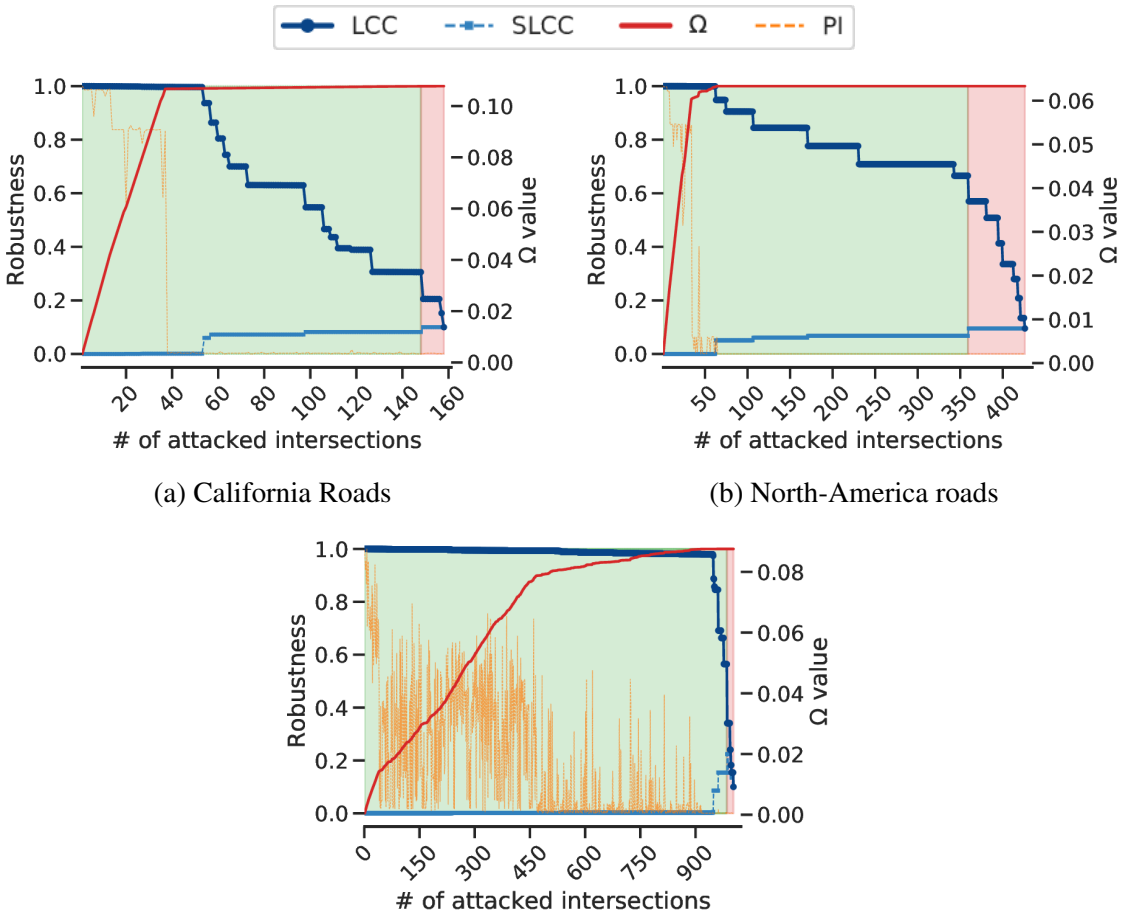


(a) Random

(b) Degree

(c) MS + R

Fig. 3.11 **Early Warning values for the SciKit European powergrid** under random failures and targeted attacks.

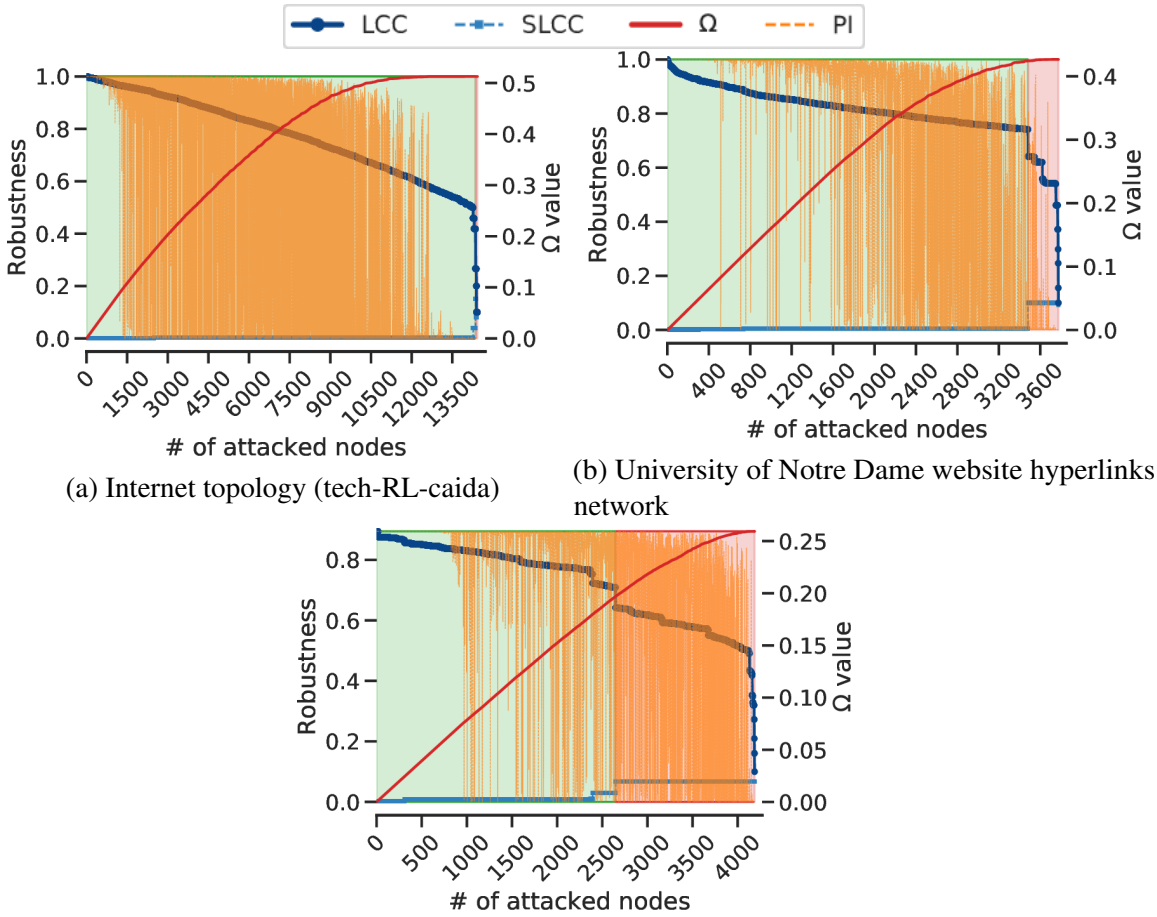


(a) California Roads

(b) North-America roads

(c) San Francisco roads

Fig. 3.12 **Ω values for three different American road networks** under *GND* +R attacks (with cost matrix $\mathbf{W} = \mathbf{I}$).



(a) Internet topology (tech-RL-caida)

(b) University of Notre Dame website hyperlinks network

(c) Stanford University website hyperlinks network

Fig. 3.13 **Ω values for three different internet networks** under GND +R attacks (with cost matrix $\mathbf{W} = \mathbf{I}$).

3.5 Understanding the models

After validating the dismantling performance of our approach, an investigation of *what* the models are actually learning and *how* they are making the long-term predictions is needed to open the black box of Deep Learning and use the resulting insights to improve the state-of-the-art algorithms. Our investigation, detailed in the following Sections, is twofold: we first attack some toy-examples to understand the model's behavior in particular situations, and then we use GNNExplainer, introduced in Section 2.5.1, to understand what sub-structures and features are captured by the models. Finally, we investigate if correlations among node features are exploited for the task.

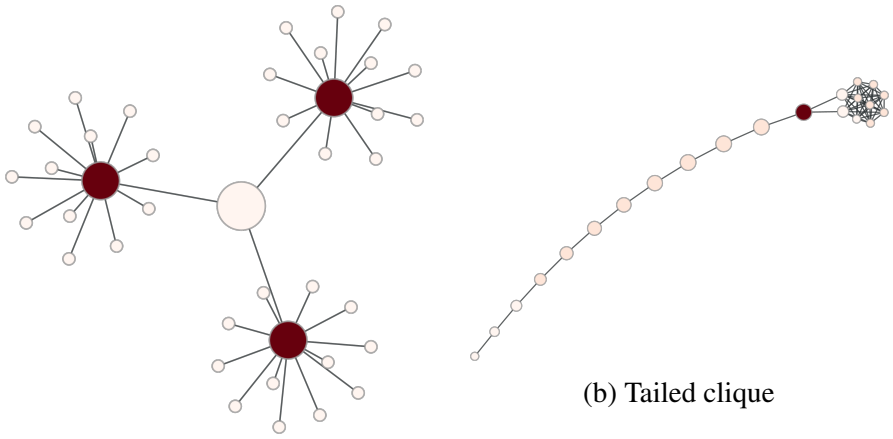
3.5.1 Models' behavior

The first part of our analysis consists in investigating the behavior of our approach by dismantling some toy-example networks. To this aim, we employ the same low computational complexity node features described before.

The first toy example, shown in Figure 3.14a, is a network built from three ego-networks joined by a bridge. The betweenness based heuristics[3], and also our common sense, would suggest removing the bridge first, reducing the LCC size to one third of the initial value, and then remove the nodes at the center of the unconnected ego networks left, for a total of four removals. Instead, our model predicts a different strategy and removes only the cores of the ego sub-networks, reaching the same LCC size with just three removals, as shown in Figure 3.15a.

At this point, we want to probe if the model is just learning to remove the nodes in descending degree order as the previous example would suggest. If that is the case, in our second toy example network, composed of a clique with an appended tail as illustrated in Figure 3.14b, the model would remove the nodes in the clique first, given their high degree. Instead, the tail is detached first, meaning that the predicted strategy differs from the degree based one, and both the degree and betweenness-based heuristics are outperformed, as shown in Figure 3.15b.

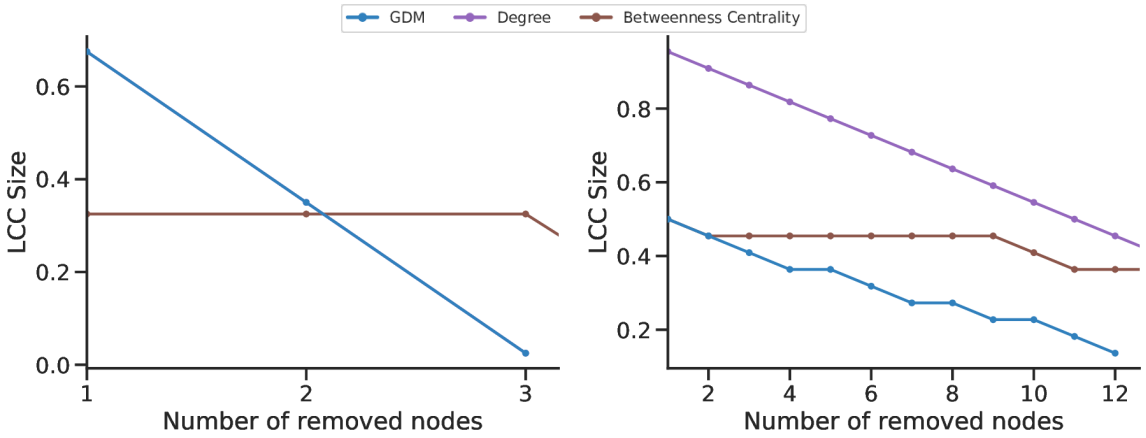
[3]The removal of nodes by descending betweenness centrality order. The node betweenness is a node centrality measure that captures the importance of the node to the shortest paths through the network.



(a) Three bridged ego networks.

(b) Tailed clique

Fig. 3.14 Toy examples. The color of the nodes represents (from dark red to white) the removal order of predicted strategy, while their size represents their betweenness value.



(a) Three bridged ego networks.

(b) Tailed clique

Fig. 3.15 Dismantling the toy example networks using our approach, GDM, and the degree and betweenness based heuristics as comparison.

3.5.2 Explaining the GNN models

The second part of the analysis consists in a deeper analysis of the models obtained by extracting the sub-structure of the graphs (and of their features) that are relevant to produce the output for each node. For this purpose, we employ GNNExplainer [171], the novel framework for explaining Graph convolutional-style networks, to extract the *explanation* sub-graphs (the sub-sets of nodes and edges) that most account for the value predicted by the model for each node.

In order to compute the explanation sub-graphs, we extend the Pytorch Geometric's implementation of *GNNExplainer* to support regression tasks by using a *Meean Square Error*

(MSE) loss function. We train the *GNNExplainer* [171] model for 600 epochs with a 0.01 learning rate, and mask out the edges with weight w lower than $E[w] + 1.5 \cdot \sigma_w$.

What we find in the analysis of the explanation sub-graphs of the networks in our test-set is that, as in the case of the *Brazilian corruption* network shown in Fig. 3.16, the model is removing the nodes that bridge multiple clusters, discovered by combining the input features and by looking to other bridges in their K-hop neighborhood, which confirms the insight provided by the toy-examples discussed the previous Section. The identification of this kind of bridges is achieved thanks to the local and second-order features combined with the propagation performed by the model. In fact, while Lauri et al. [13] show that the degree, its χ^2 value and the local clustering coefficient can be used to estimate the likelihood a node belongs to a clique via classical Deep Learning tools, our Geometric Deep Learning model improves the idea by extending the feature propagation in a K-hop radius and the result is improved further by the k–core value that helps to filter the nodes at the core of the network. Although some of the targeted nodes are not the direct cause of large damage to the network, they are needed to drive the network in a vulnerable state where the removal of other nodes disrupts it. In other words, the models seem to predict a long-term strategy that aims not only to remove the *Articulation Points* (AP, also known as Cut Vertices, are nodes that, when removed, cause the creation of a new connected component) but also create new ones with the removal of other non-AP nodes.

This insight led us to investigate further in this direction with an analysis of the Articulation Points as the nodes are removed. Specifically, we compute, removal after removal, the number of APs in the network and how many of them are in the removal list (R) predicted by the model.

As shown in Figures 3.17a and 3.17b for the *linux* and *internet-topology* networks, the number of APs increases as nodes are removed, and so do the ones in the removal list, until there is a natural decay due to the decreasing size of the removal list itself. This trend is confirmed for most of our the 45 real-world test networks, as shown in Figure 3.19.

Considering the high dismantling performance discussed in the previous sections, this proves that not only the model is effectively learning to target the nodes that cause the network collapse when removed together, but also that does so more efficiently than other algorithms. Note that a strategy barely based on AP removal would not be effective, since an AP can be one node whose removal separates a giant connected component from a component consisting of a negligible number of nodes (e.g., only one node). Instead, we demonstrate that our model is learning to identify the most effective AP for disintegrating the target system: elegantly, these turn out to be bridges between large clusters, not between one large and one small cluster.

Moreover, if we analyze the number of APs in the removal list ($|AP \cap R|$) as a function of the total number of APs ($|AP|$), we find that the two are related by a kind of deterministic dynamics, resembling the one which characterizes chaotic systems and, specifically, chaotic maps such as the logistic map or the Hénon map, where parabolic attractors emerge when the state of the system at the $n+1$-th step is plotted against the state at the n-th step. In our case, the n-th step coincides with the removal of the n-th node in the removal list. The shape of the resulting attractor provides a strong characterization of the system and its robustness: we show an example for each type in Figures 3.17c and 3.17d. More examples can be found in Figure 3.20. That is, in the first case, the model drives the network in a state where the nodes in the removal list become Articulation Points, in the latter it mainly removes nodes that are already APs.

After understanding *what* the model is learning, we analyze how features account in the computation of the output values to get an insight on *how* the model selects the nodes. While there is no prevailing feature for all the networks — e.g., sometimes the degree is the key feature, others the K–core value, etc. — an interesting result is that the feature weight also changes with the score of the nodes. For instance, while the clustering coefficient is the main feature, scoring up to the 60% of the relative importance, in the first 250 removals of the *subelj-jdk* network (Figure 3.17f), all the features gain equal weight after that removal. In the *Brazilian corruption* network, instead, the node degree is the most important feature to identify the first nodes to remove, but other features gain more importance to identify less important nodes, needed to reach the dismantling target. These results confirm that the definition of new algorithms based on these insights is extremely hard, as the weight of each feature is adapted by the model to the topology and to the patterns in the network. At this point, it is plausible to assess that our framework learns correlations among node features. To probe this hypothesis, in Section 3.5.3 and in Fig. 3.21, we analyze the configuration models of the same networks analyzed so far: those models keep the observed connectivity distribution while destroying topological correlations. We observe that dismantling performance drop on these models, confirming that the existing topological correlations are learned and, consequently, exploited by the machine. For more details, we refer the Reader to the dedicated Section 3.5.3.

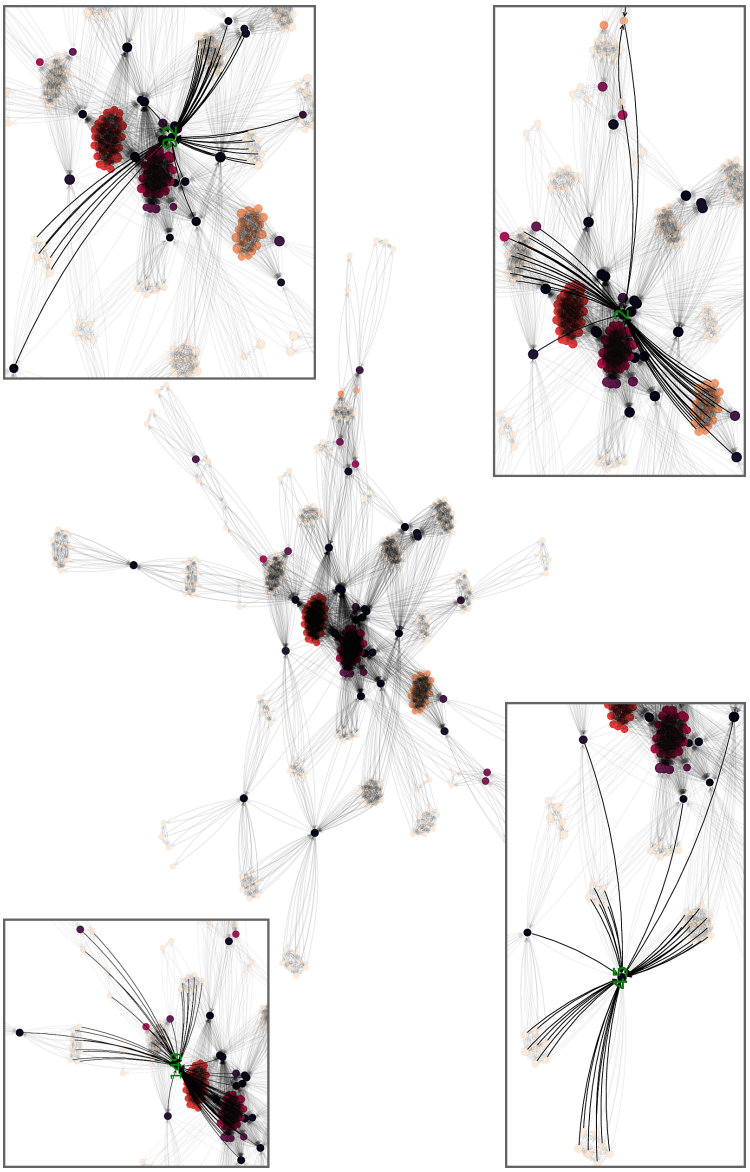
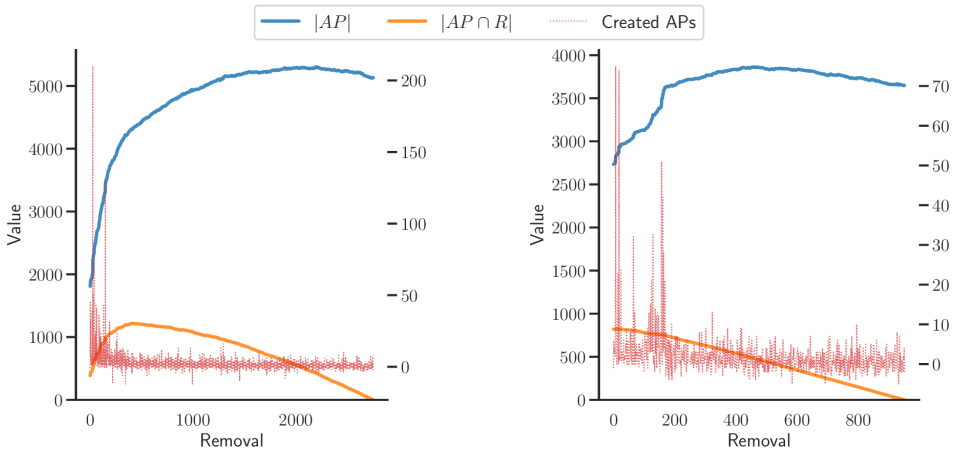
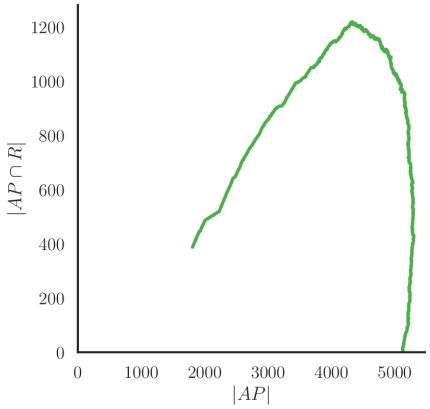


Fig. 3.16 **Explanation sub-graphs** for the first four nodes of the *Brazilian corruption* network. The model is targeting nodes that act as bridge between multiple clusters and the choice is also based on neighboring nodes that are bridges themselves.

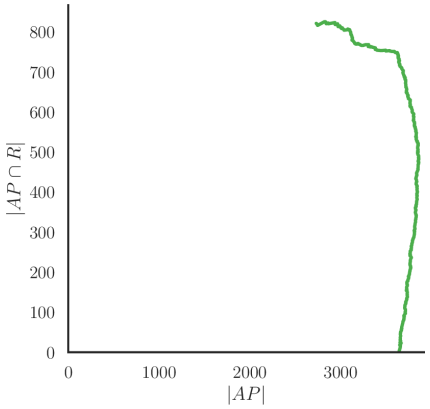


(a) *linux*: *AP* trend.

(b) *internet-topology*: *AP* trend.



(c) *linux*: *AP* and *R* relation.

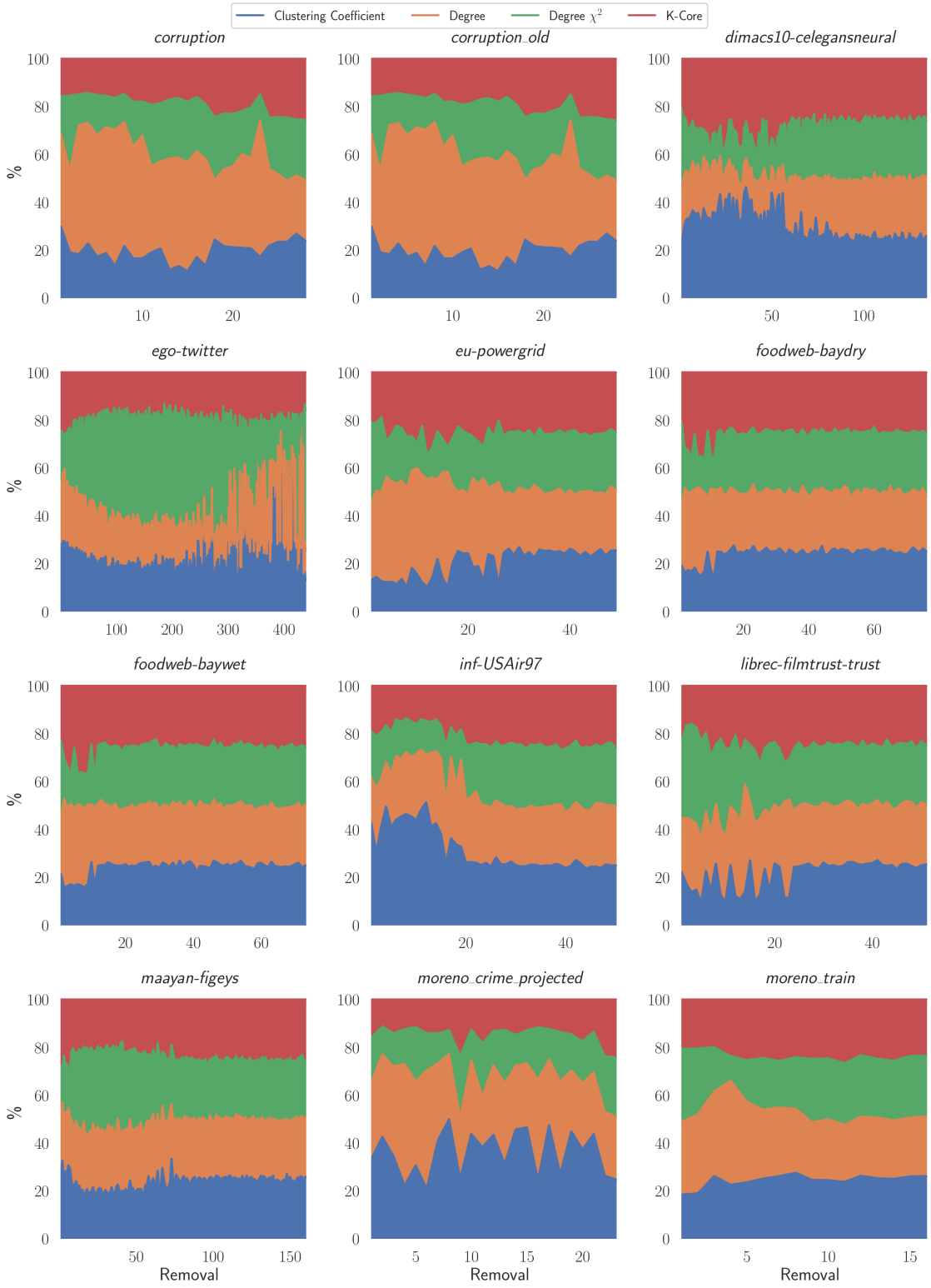


(d) *internet-topology*: *AP* and *R* relation.

(e) *corruption*: features importance trend.

(f) *subelj-jdk*: features importance trend.

Fig. 3.17 **Understanding our models.** The analysis of the Articulation Points of the networks (*AP*) and how many of them are in the removal list (*R*) shows that the models are learning a long-term strategy that aims to create new articulation points and remove the ones that deal most damage to the network. This is achieved using the input node features discussed above, that allow the identification of clusters and bridges.


Clustering Coefficient
Degree
Degree χ^2
K-Core
corruption
corruption_old
dimacs10-celegansneural
ego-twitter
eu-powergrid
foodweb-baydry
foodweb-baywet
inf-USAir97
librec-filmtrust-trust
maayan-figeys
moreno_crime_projected
moreno_train
%
Removal

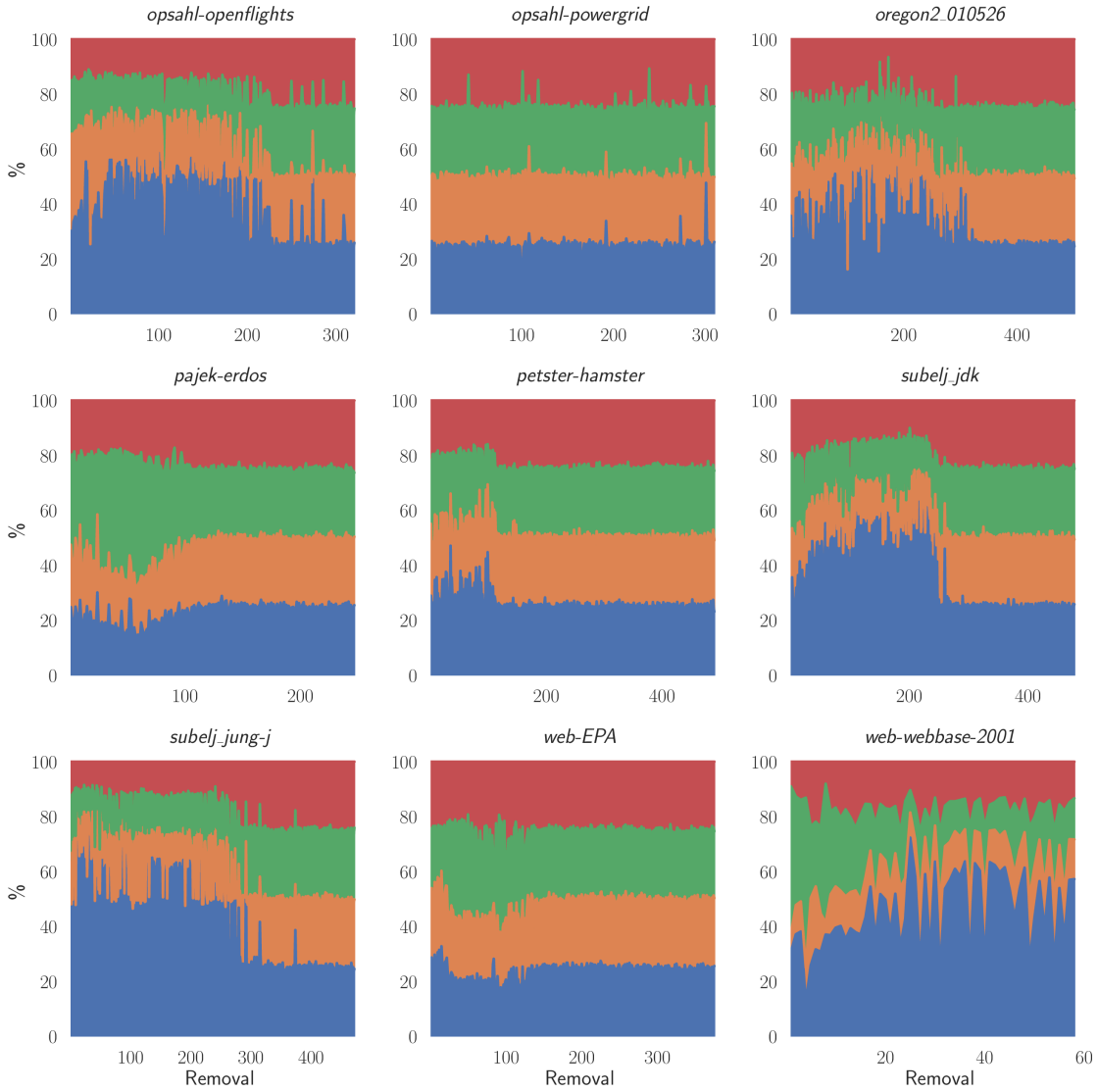
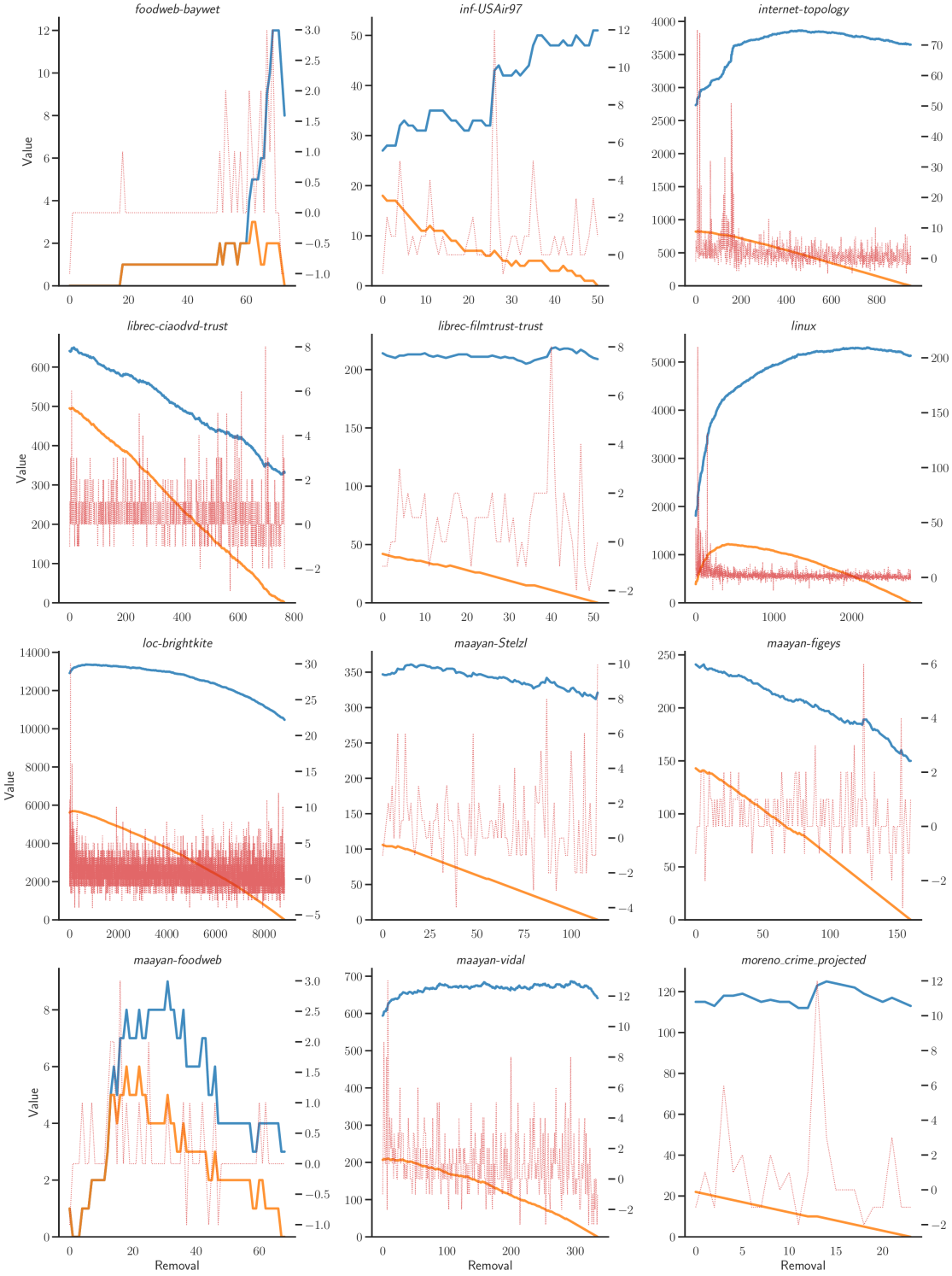
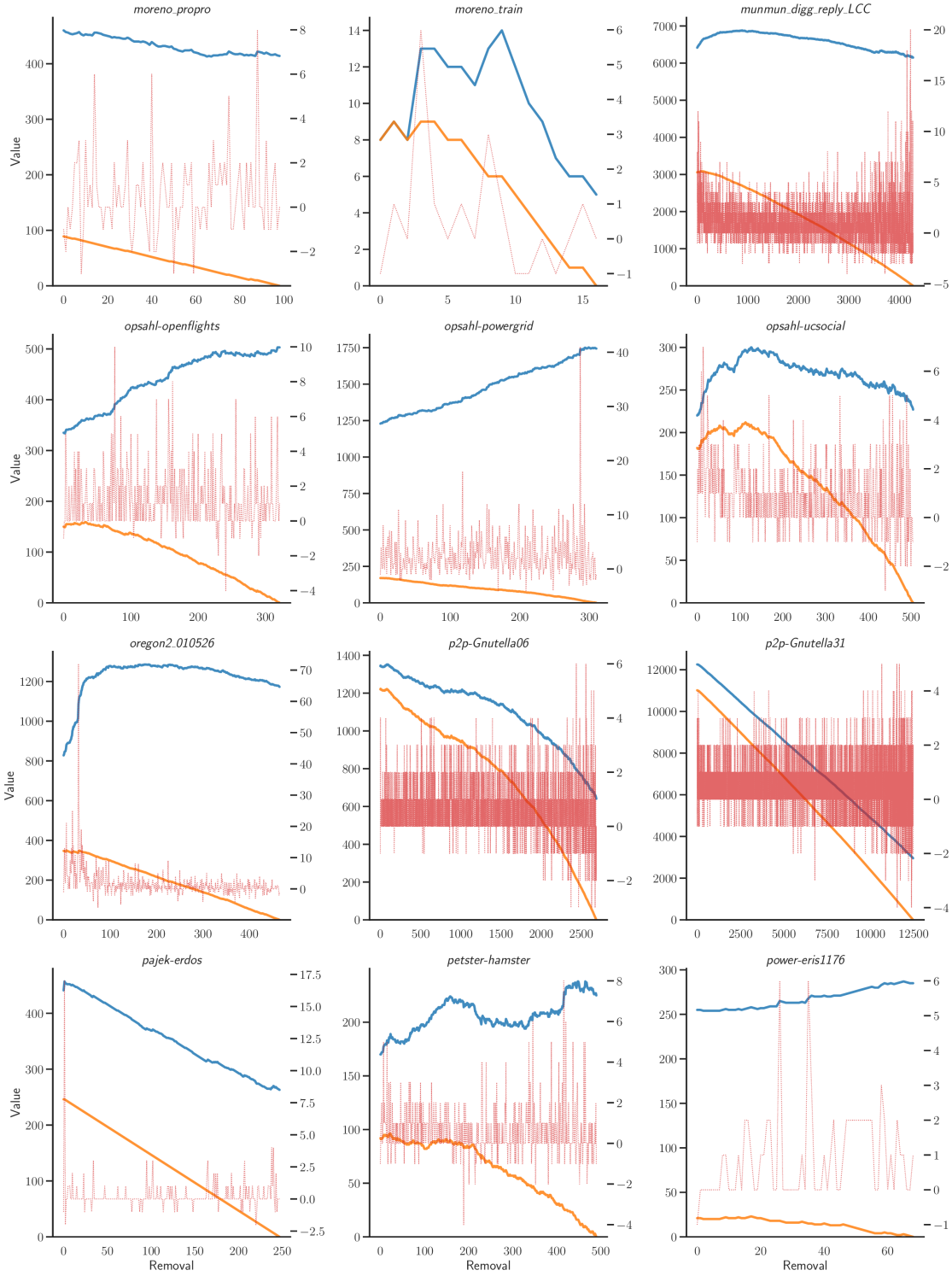


Fig. 3.18 **Features' importance trend.** Relative features' importance in the computation of each p_n value, provided by *GNNExplainer*, in removal order.


|AP|
|AP ∩ R|
Created APs
ARK201012.LCC
advogato
arenas-meta
cfinder-google
corruption
dblp-cite
dimacs10-celegansneural
dimacs10-polblogs
econ-wm1
ego-twitter
eu-powergrid
foodweb-baydry
Value
Removal


foodweb-baywet
inf-USAir97
internet-topology
librec-ciaodvd-trust
librec-filmtrust-trust
linux
loc-brightkite
maayan-Stelzl
maayan-figeys
maayan-foodweb
maayan-vidal
moreno_crime_projected
Value
Removal


moreno_propro
moreno_train
munmun_digg_reply_LCC
opsahl-openflights
opsahl-powergrid
opsahl-ucsocial
oregon2_010526
p2p-Gnutella06
p2p-Gnutella31
pajek-erdos
petster-hamster
power-eris1176
Value
Removal

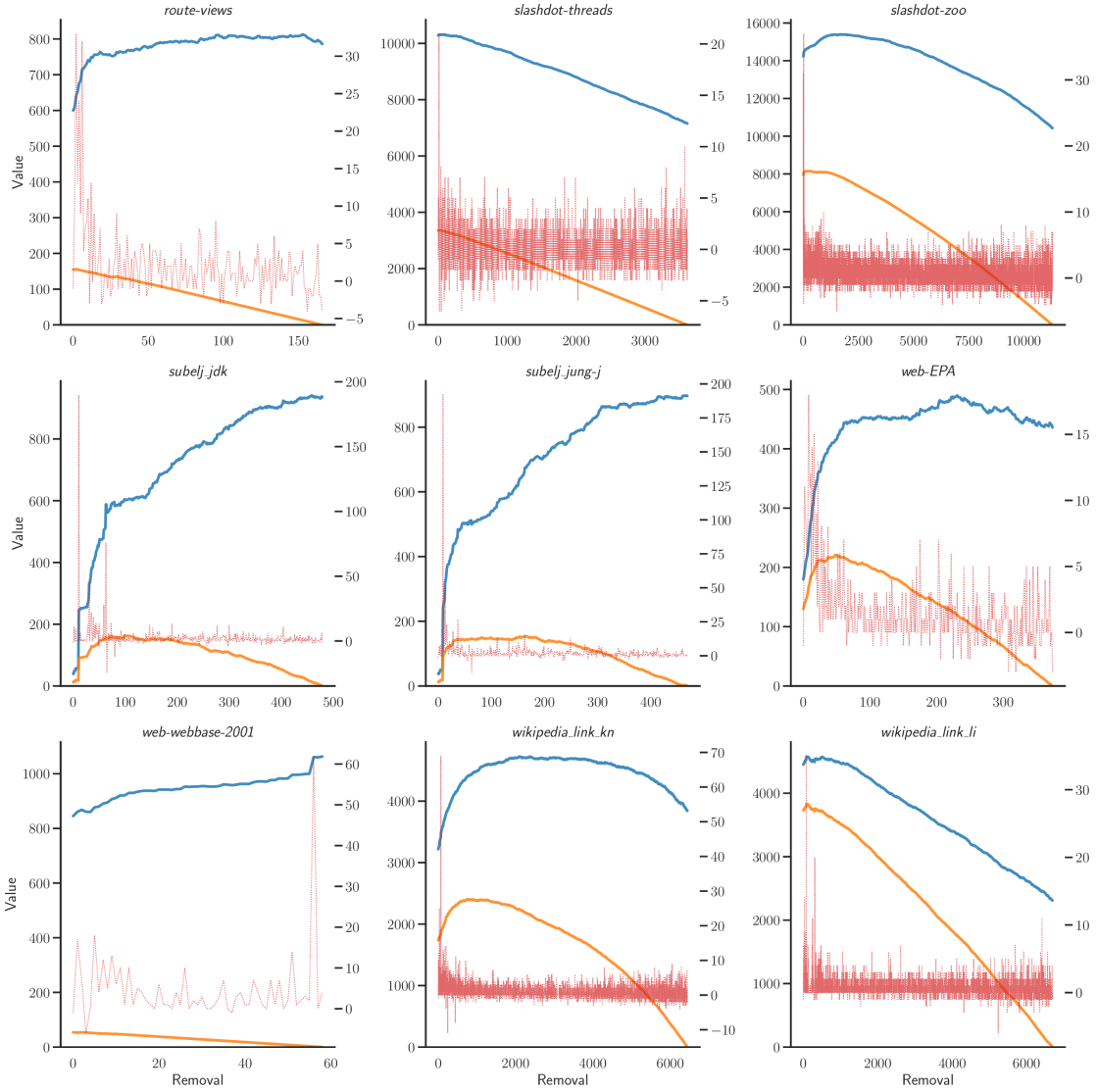
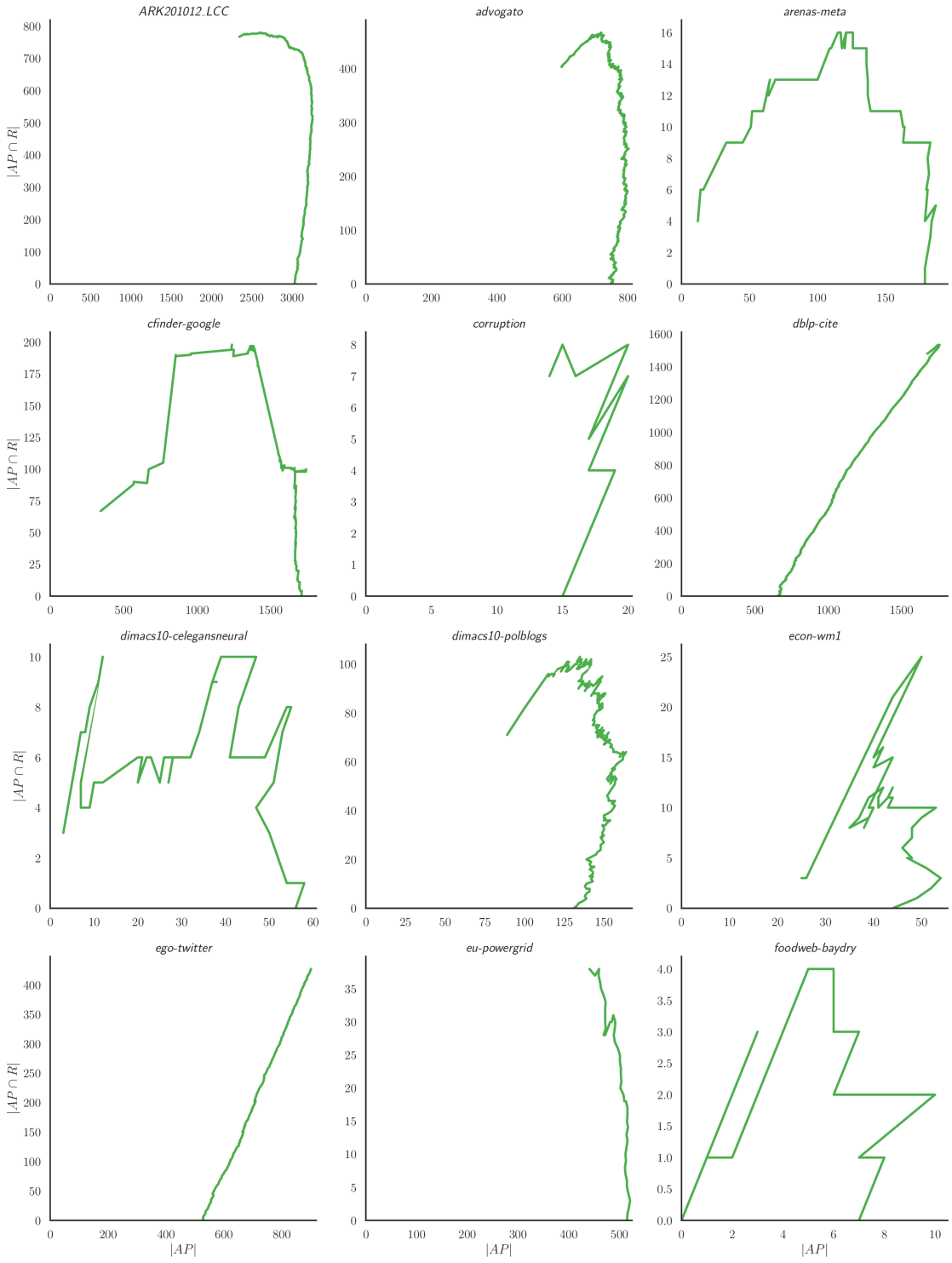
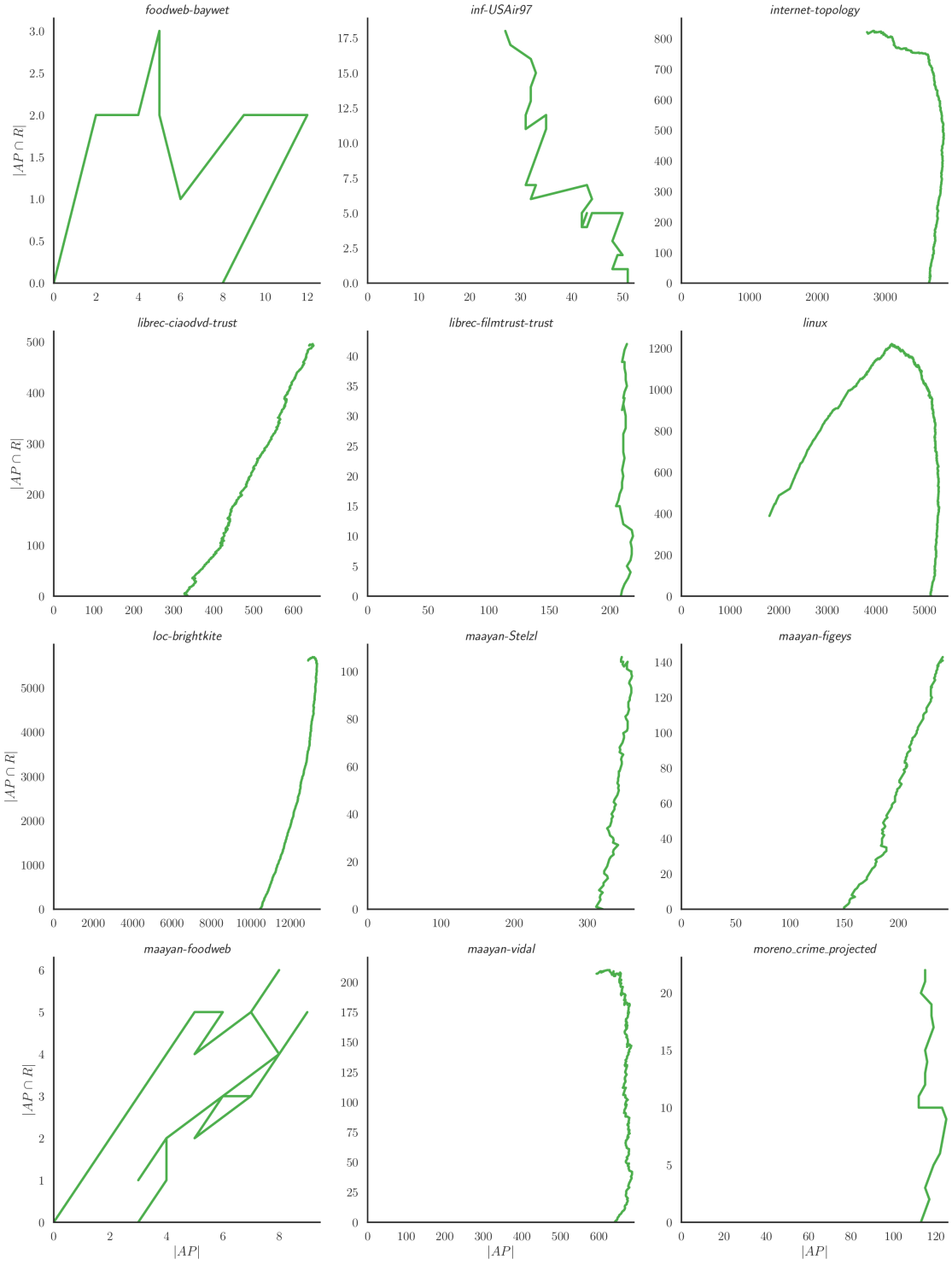
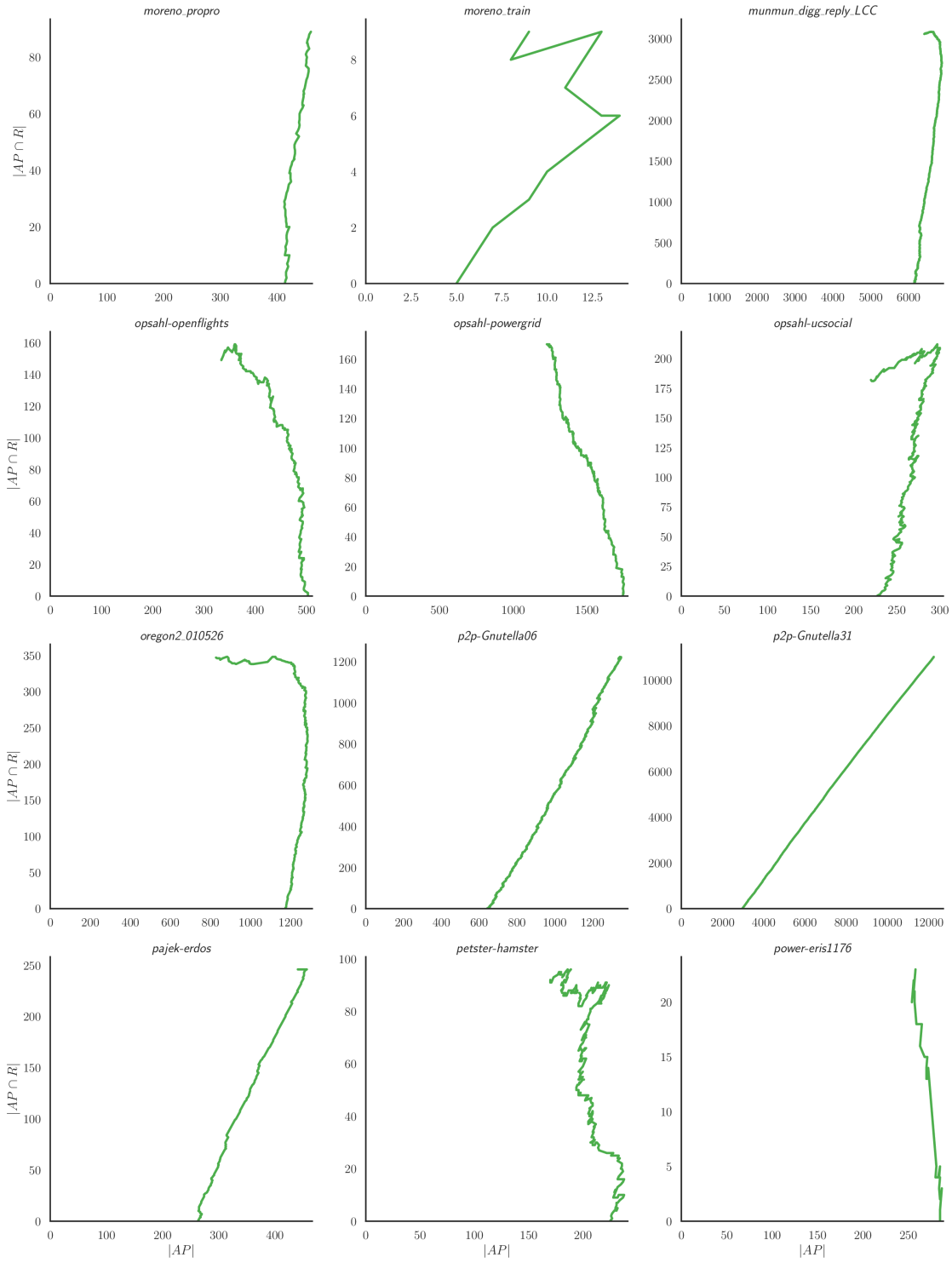


Fig. 3.19 **Articulation Point trend.** We compute, removal after removal, the number of APs in the network ($|AP|$), the number of APs in the removal list ($|AP \cap R|$) and the number of created APs.


ARK201012.LCC
advogato
arenas-meta
cfinder-google
corruption
dblp-cite
dimacs10-celegansneural
dimacs10-polblogs
econ-wm1
ego-twitter
eu-powergrid
foodweb-baydry
$|AP \cap R|$
$|AP|$


foodweb-baywet
inf-USAir97
internet-topology
librec-ciaodvd-trust
librec-filmtrust-trust
linux
loc-brightkite
maayan-Stelzl
maayan-figeys
maayan-foodweb
maayan-vidal
moreno_crime_projected
$|AP \cap R|$
$|AP|$


moreno_propro
moreno_train
munmun_digg_reply_LCC
opsahl-openflights
opsahl-powergrid
opsahl-ucsocial
oregon2_010526
p2p-Gnutella06
p2p-Gnutella31
pajek-erdos
petster-hamster
power-eris1176
$|AP \cap R|$
$|AP|$

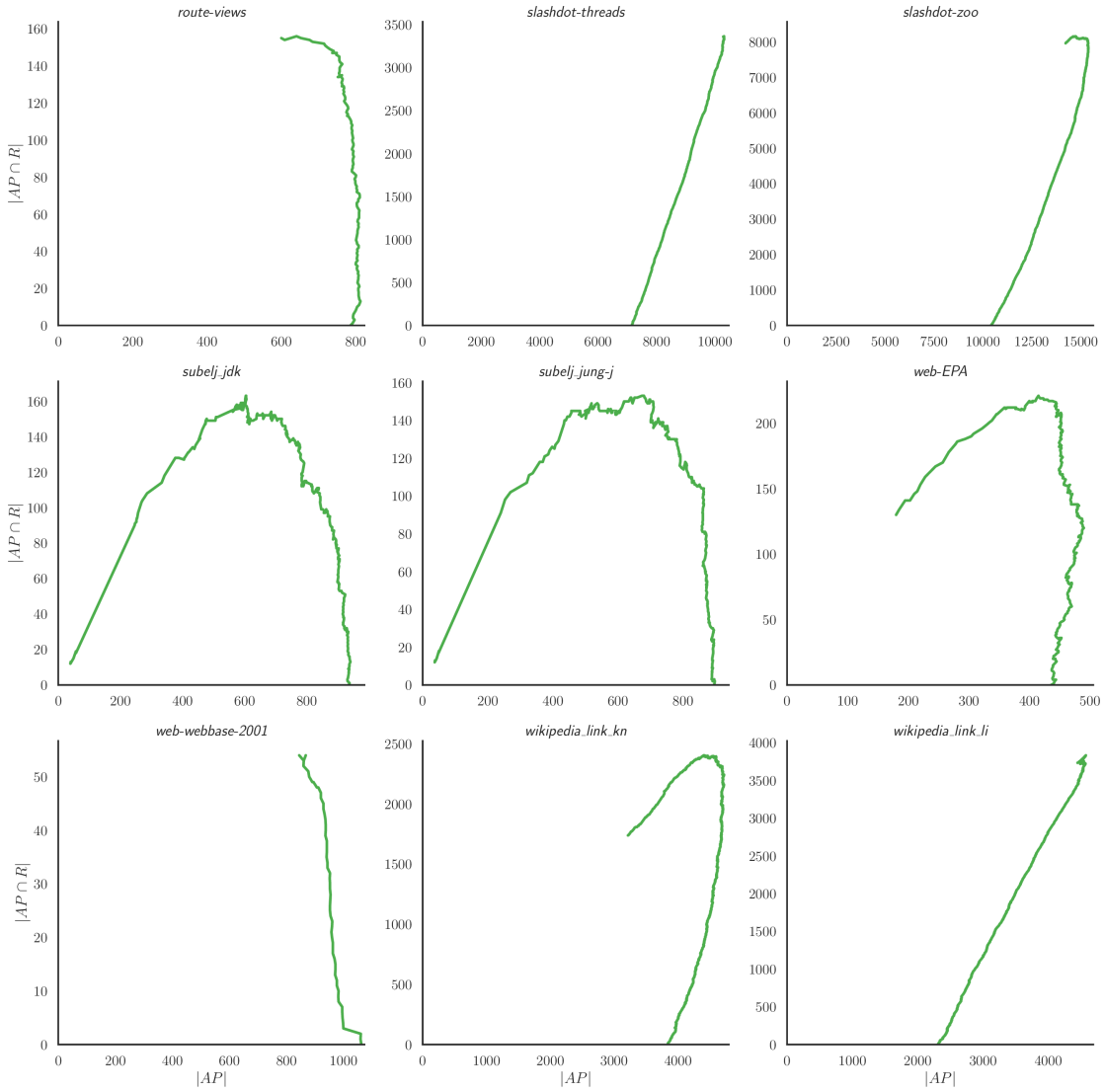
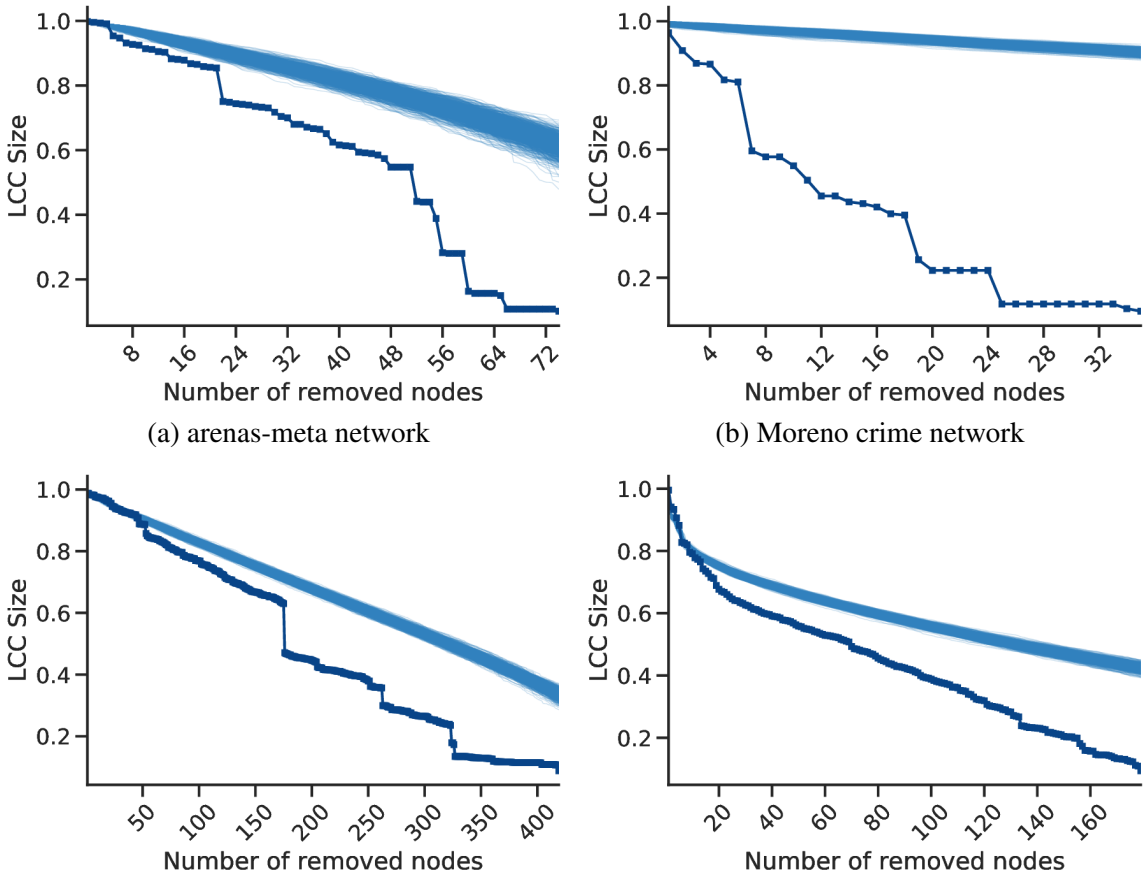


Fig. 3.20 **Relation between the number of APs and the number of APs in the removal list.** The two are related by a kind of deterministic dynamics, resembling the one which characterizes chaotic systems and, specifically, chaotic maps such as the logistic map or the Hénon map, where parabolic attractors emerge when the state of the system at the $n+1$-th step is plotted against the state at the n-th step. In our case, the n-th step coincides with the removal of the n-th node in the removal list. The shape of the resulting attractor provides a strong characterization of the system and its robustness.

3.5.3 Dismantling of configuration model rewired networks

We investigate further if the model is learning correlations among node features by dismantling the configuration model rewirings[4] of the networks in our test set. If that is the case, the dismantling power of our approach on the rewirings should be heavily affected. In Figure 3.21 we show, for each network, the dismantling of 1000 configuration models and also the original instance as comparison. In all the tested networks, there is a severe performance drop. For instance, in Figures 3.21b it takes just ~ 35 removals to dismantle the original instance of the Moreno crime network, while the LCC size of the rewired networks after the same number of removals is still very large (i.e., $\sim 95\%$). This result confirms our insight. That is, existing topological correlations are learned and, consequently, exploited by the machine.

[4]The configuration model of a network keeps the observed connectivity distribution while destroying topological correlations, meaning that feature correlations are lost.



(a) arenas-meta network

(b) Moreno crime network

(c) opsahl-openflights network

(d) route-views network

Fig. 3.21 **Dismantling of configuration model rewirings** (light blue, 1000 per network) and of the original networks (dark blue).

3.6 Computational complexity

The computational complexity of our approach mainly depends on two elements: 1) the computational complexity of the node features used and 2) the computational complexity of the convolutional-style layers in the model. In particular, the convolutional-style layers that we employ, i.e., the *Graph Attention Networks*, scale as $O(N+E)$ where N is the number of nodes and E is the number of edges in the network. Considering that real-world networks are usually sparse, we assume that $O(E) \approx O(N)$, so $O(N+E) \approx O(N)$, and the computational complexity of our approach is the maximum between this and the computational complexity of the features. Given that, the most expensive feature we compute in our experiments is the k–coreness, that is $O(N+E)$, so the computational complexity of the approach detailed above is $O(N)$. For what concerns the computational complexity of the brute-force performed during the training set generation, it is irrelevant as it is a highly parallelizable one-time task that is performed on very small networks. Moreover, since the neural models can generalize, there is no need to train them for each dismantling, and the actual time spent training is negligible.

3.7 Discussion

Our results show that using machine learning to learn network dismantling comes with a series of advantages. While the ultimate theoretical framework is still missing, our framework allows one to learn directly from the data, at variance with traditional approaches which rely on the definition of new heuristics, metrics or algorithms. An important advantage of our method, typical of data-driven modeling, is that it can be further improved by simply retuning the parameters of the underlying model and training again: conversely, existing approaches require the (re)definition of heuristics and algorithms which are more demanding in terms of human efforts. Remarkably, the computational complexity of dismantling networks with our framework is considerably low: just $O(N+E)$, where N is system's size and E the number of connections — which drops to $O(N)$ for sparse networks (for more information about the computational complexity, see Section 3.6). This feature allows for applications to systems consisting of millions of nodes while keeping excellent performance in terms of computing time and accuracy. We also provide deep-insights about the models that should help to understand the power of Geometric Deep Learning. Last but not least, from a methodological perspective, it is worth remarking that our framework is general enough to be adapted and applied to other interesting NP-hard problems on networks, opening the door

for new opportunities and promising research directions in complexity science, together with very recent results employing machine learning, for instance, to predict extreme events [220].

The impact of our results is broad. On the one hand, we provide a framework[5] which disintegrates real systems more efficiently and faster than state-of-the-art approaches: for instance, applications to covert networks might allow to hinder communications and information exchange between harmful individuals. On the other hand, we provide a quantitative descriptor of damage which is more predictive than existing ones, such as the size of the largest connected component: our measure allows to estimate the potential system's collapse due to subsequent damages, providing policy and decision makers with a quantitative early-warning signal for triggering a timely response to systemic emergencies in water management systems, power grids, communication and public transportation networks.

3.8 Dataset

In Table 3.5 we list the test networks used in our experiments with their category and size (number of nodes and edges). Those networks model systems from various domains (e.g., biological, infrastructure and social data and so on), and range from a few hundred of nodes to more than one million. For more about each network, we refer the Reader to the original source.

[5]The code of our framework is available on GitHub at https://github.com/NetworkScienceLab/GDM and on Zenodo at https://doi.org/10.5281/zenodo.5105911 [221]

Network	Name	Category	$\|N\|$	$\|E\|$	References
ARK201012_LCC	CAIDA ARK (Dec 2010) (LCC)	Infrastructure	29.3K	78.1K	[222]
advogato	Advogato trust network	Social	6.5K	43.3K	[223, 224]
arenas-meta	C. elegans	Metabolic	453	2.0K	[225, 226]
cfinder-google	Google.com internal	Hyperlink	15.8K	149.5K	[227, 228]
citeseer	CiteSeer	Citation	384.4K	1.7M	[229, 230]
com-dblp	DBLP co-authorship	Coauthorship	317.1K	1.0M	[231, 232]
corruption	Corruption Scandals	Social	309	3.3K	[218]
dblp-cite	DBLP citation	Citation	12.6K	49.6K	[233, 234]
digg-friends	Digg friends	Social	279.6K	1.5M	[235, 236]
dimacs10-celegansneural	C. elegans (neural)	Neural	297	2.1K	[237, 48, 238]
dimacs10-polblogs	Political blogs (LCC)	Hyperlink	1.2K	16.7K	[239, 240]
douban	Douban social network	Social	154.9K	327.2K	[241, 242]
econ-wm1	Economic network WM1	Economic	260	2.6K	[243]
ego-twitter	Twitter lists	Social	23.4K	32.8K	[244, 245]
email-EuAll	EU institution email	Communication	265.2K	365.6K	[246, 247]
eu-powergrid	SciGRID Power Europe	Power	1.5K	1.8K	[248]
foodweb-baydry	Florida ecosystem dry	Trophic	128	2.1K	[249, 250]
foodweb-baywet	Florida ecosystem wet	Trophic	128	2.1K	[251, 250]
gridkit-eupowergrid	GridKit Power Europe	Power	13.8K	17.3K	[252]
gridkit-north_america	GridKit Power North-America	Power	16.2K	20.2K	[252]
hyves	Hyves social network	Social	1.4M	2.8M	[253, 242]
inf-USAir97	US Air lines (1997)	Infrastructure	332	2.1K	[254, 243, 255]
internet-topology	Internet (AS) topology	Infrastructure	34.8K	107.7K	[256, 257]
librec-ciaodvd-trust	CiaoDVD trust network	Social	4.7K	33.1K	[258, 259]
librec-filmtrust-trust	FilmTrust trust network	Social	874	1.3K	[260, 261]
linux	Linux source code files	Software	30.8K	213.7K	[262]
loc-brightkite	Brightkite friendships	Social	58.2K	214.1K	[263, 264]
loc-gowalla	Gowalla friendships	Social	196.6K	950.3K	[265, 264]
london_transport_multiplex_aggr	Aggregated London Transportation network	Transport	369	430	[266]
maayan-Stelzl	Human protein (Stelzl)	Metabolic	1.7K	3.2K	[267, 268]
maayan-figeys	Human protein (Figeys)	Metabolic	2.2K	6.4K	[269, 270]
maayan-foodweb	Little Rock Lake food web	Trophic	183	2.5K	[271, 272]
maayan-vidal	Human protein (Vidal)	Metabolic	3.1K	6.7K	[273, 274]
moreno_crime_projected	Crime (projection)	Social	754	2.1K	[275]
moreno_propro	Protein	Metabolic	1.9K	2.3K	[276–279]
moreno_train	Train bombing terrorist contacts	Human contact	64	243	[280, 281]
munmun_digg_reply_LCC	Digg social network replies (LCC)	Communication	29.7K	84.8K	[282, 283]
munmun_twitter_social	Twitter follows (ICWSM)	Social	465.0K	833.5K	[284, 285]
opsahl-openflights	OpenFlights	Infrastructure	2.9K	15.7K	[286, 287]
opsahl-powergrid	US power grid	Infrastructure	4.9K	6.6K	[288, 48]
opsahl-ucsocial	UC Irvine messages	Communication	1.9K	13.8K	[289, 290]
oregon2_010526	Autonomous systems Oregon-2	Infrastructure	11.5K	32.7K	[291]
p2p-Gnutella06	Gnutella P2P, August 8 2002	Computer	8.7K	31.5K	[292, 293]
p2p-Gnutella31	Gnutella P2P, August 31 2002	Computer	62.6K	147.9K	[294, 292]
pajek-erdos	Erdős co-authorship network	Coauthorship	6.9K	11.8K	[295, 255]
petster-catdog-household	Catster/Dogster familylinks (LCC)	Social	324.9K	2.6M	[296]
petster-hamster	Hamsterster full	Social	2.4K	16.6K	[297]
power-eris1176	Power network problem	Power	1.2K	9.9K	[243]
roads-california	California Road Network	Infrastructure	21.0K	21.7K	[298]
roads-northamerica	North-America Road Network	Infrastructure	175.8K	179.1K	[299]
roads-sanfrancisco	San Francisco Road Network	Infrastructure	175.0K	221.8K	[300]
route-views	Autonomous systems AS-733	Infrastructure	6.5K	13.9K	[301, 247]
slashdot-threads	Slashdot threads	Communication	51.1K	117.4K	[302, 303]
slashdot-zoo	Slashdot Zoo	Social	79.1K	467.7K	[304, 305]
subelj_jdk	JDK dependency network	Software	6.4K	53.7K	[306]
subelj_jung-j	JUNG and Javax dependency network	Software	6.1K	50.3K	[307, 308]
tech-RL-caida	Internet router network	Infrastructure	190.9K	607.6K	[243]
twitter_LCC	Twitter users (LCC)	Social	532.3K	694.6K	[202]
web-EPA	Pages linking to epa.gov	Hyperlink	4.3K	8.9K	[309, 243]
web-NotreDame	Notre Dame web pages	Hyperlink	325.7K	1.1M	[310, 69]
web-Stanford	Stanford University web pages	Hyperlink	281.9K	2M	[311, 312]
web-webbase-2001	Web network	Hyperlink	16.1K	25.6K	[313, 314, 243]
wikipedia_link_kn	Wikipedia links (KN)	Hyperlink	29.5K	278.7K	[315]
wikipedia_link_li	Wikipedia links (LI)	Hyperlink	49.1K	294.3K	[316]
wordnet-words	WordNet lexical network	Lexical	146.0K	657.0K	[317, 318]

Table 3.5 The networks used to evaluate our approach. For each network, we report the name, the number of nodes and edges, the category it belongs to and some references.

3.9 Test environment

Here we detail the environment where our experiments were performed and the tools used.

All experiments ran on a shared machine equipped with two Intel Xenon E5-2620 CPUs, 128GB RAM and a two core nVidia Tesla K80 (with 12GB VRAM each). More details about the drivers used and the full package dependency list of our code can be found in the code package.

Concerning the other algorithms used in our comparison (i.e., *GND*, *EGND MS*, *CoreHD* and *EI*), we use authors' official code with default parameters. Specifically, we use identity weight input matrix for both *GND* and *EGND* (and the relative fine-tuning algorithm), $1K$ trials for the *EGND*.

3.10 Appendix: Network Dismantling exploiting network geometry

In this Section we briefly introduce the preliminary research conducted during my remote internship at the Center for Complex Network Intelligence (CCNI) at Tsinghua University, Beijing, China, with the additional supervision of Dr. Carlo Vittorio Cannistraci, the Director of the Center.

3.10.1 Introduction

In the previous Sections we outlined how Geometric Deep Learning can be used to learn the problem of dismantling complex systems. In particular, after demonstrating the dismantling performance on empirical complex systems, we showed in Section 3.5 that the models are first identifying as high-priority targets those nodes that bridge between clusters. That is, removing particular bridge nodes is a viable strategy to deal high damage to the target system.

During my internship, we aimed at exploiting this insight to design a non-learnable algorithm to identify such nodes. Specifically, the idea is to leverage the Repulsion-Attraction (RA) algorithms [319], that assign an edge weight using the local information about its adjacent nodes, to produce node scores based on network geometry and to remove nodes according to such score.

3.10.2 Formulation and Preliminary results

We employ a variant of the Repulsion-Attraction (RA1) score, formally defined as in equation 3.3 [319], that has motivations rooted in the theory of navigability [320] and uses local topological information about the adjacent node's neighborhoods to assign a weight RA^1_{L2} to each edge. More in detail, the $\mathrm{RA}^1_{L2}(i,j)$ of edge between nodes i and j accounts for the number of external nodes (i.e., those nodes that do not belong to the common neighborhood of the adjacent nodes), and for the common neighbors. The former contribute to the *repulsion* term, and the latter to the *attraction* one.

$$\mathrm{RA}^1_{L2}(i,j) = \frac{1+\mathrm{eRA1}_{L2}(i,j)}{1+|\mathrm{CN2}_{ij}|} \tag{3.3}$$

where $\mathrm{eRA1}_{L2}(i,j) = |E2_{ij}| + |E2_{ji}|$ with $E2_{ij} = \mathscr{N}_i - \mathrm{CN2}_{ij}$, $\mathrm{CN2}_{ij}$ being the external neighborhood of i with j.

The RA values are then aggregated at node-level using two different functions (sum and max) to get a score that can be used to rank the nodes, and define:

$$\mathrm{RA}^1_{L2}\mathrm{Sum}(i) = \sum_j^{\mathscr{N}_i} \mathrm{RA}^1_{L2}(i,j) \tag{3.4}$$

$$\mathrm{RA}^1_{L2}\mathrm{Max}(i) = \max_j^{\mathscr{N}_i} \mathrm{RA}^1_{L2}(i,j) \tag{3.5}$$

In addition, we also extend the RA formulation to consider the L3 (2-hop) neighborhood of each node.

We validate these heuristics on 23 of the networks used to benchmark GDM and detailed in Section 3.8, and report the preliminary results in Table 3.6. More in detail, while, on average, the RAs do not outperform GDM, it is interesting to note that some formulations do with a significant margin in some networks (e.g., RA1 L3 Sum scores almost $\sim 10\%$ less AUC in the librec-filmtrust-trust network). These promising results suggest that there is a margin of improvement, for instance by attacking the k–core of the network or by removing only nodes that belong to the Largest Connected Component of the network.

Heuristic	GDM	RA1 L2 Sum (D)	RA1 L3 Sum (D)	RA1 L2 Max (D)	RA1 L3 Max (D)
arenas-meta	100	102.2	204.2	122.1	208.8
corruption	100	141.8	497.8	626.9	785.7
de-powergrid	100	150.4	119.9	152.8	175.1
dimacs10-celegansneural	100	99.2	105.3	122.9	125.6
dimacs10-polblogs	100	98.7	94.5	111.6	110.3
econ-wm1	100	104.4	105.7	115.8	115.7
eu-powergrid	100	154.1	125.2	140.1	158.8
foodweb-baydry	100	97.7	99.2	118.1	118.9
foodweb-baywet	100	98.9	101.2	121.1	122.2
inf-USAir97	100	113.1	107.6	111.8	125.5
librec-filmtrust-trust	100	92.5	90.7	128.8	125.9
london_transport_multiplex_aggr	100	109.1	106.2	140.6	141.1
maayan-foodweb	100	113.3	113.3	206.7	170.8
maayan-vidal	100	108.8	96.3	161.4	116.3
moreno_crime_projected	100	137.4	134.4	272.8	325.4
moreno_propro	100	109.3	105.6	135.4	124.3
moreno_train	100	107.7	99.1	157.4	158.4
opsahl-openflights	100	122.1	104.2	134.4	119.7
opsahl-powergrid	100	130.6	129.8	157.6	167.3
opsahl-ucsocial	100	98.1	95.4	108.3	120
petster-hamster	100	90.1	110.1	105.4	133.9
power-eris1176	100	150.9	472	370.9	349.6
web-EPA	100	101.2	97.9	115.5	115.1
Average	100	114.4	144.2	171.2	183.2

Table 3.6 Preliminary Repulsion-Attraction (RA) results. The Table shows the preliminary results on a sub-set of 23 real-world networks.

Chapter 4

Learning the Link Building Problem

4.1 Introduction

The increased computational capabilities opened new frontiers to the study of large scale phenomena and also allows researchers to adopt a different point of view to analyze traditional problems. From this perspective, the use of network representation to model interactions among several entities has quickly grown [321], and helps in understanding and managing different types of network-based phenomena, as in computer communications, transport infrastructures, online social systems, metabolic reactions, financial markets and many others [322].

In many of these application scenarios, an important question is how to assess the importance of a node or, in other words, how to define a measure of its centrality. The definition of centrality measures is an important topic in the field of network research due to its various theoretical and practical implications. There are different measures of centrality depending on the point of view, the context and the meaning of importance that we want to attribute to a node [323]. For instance, if we are looking for brokers then *betweenness centrality* [52, 324] could be the best metric, whereas if the target is prestige the *in-degree* can be used, while *out-degree* is a good metric for activity. The eigenvector-based approach *PageRank* [57] is a widely used metric for powerful collaborative partners centrality [325].

In general, all the centrality measures try to answer to the following basic question: *how to rank the nodes (whatever the node represents) according to their position in the network?* When rank is used, regardless of the reason, the improvement of the position is also a topic of large interest. In fact, a better positioning of the node in the network, with respect to a specific centrality metric, corresponds to a greater importance of the node in the modeled system, which translates, in many real-world contexts, into some form of advantage over the other nodes.

For example, Google used to rank the web pages via the PageRank metric to determine the order in which the results of a search were displayed. In a social context, the centrality of a node can be used to determine the ability of a person to influence others with their own opinions. In the international trade exchanges, a country that is more *central* than others plays a role of strategic importance in the market. It is therefore natural that in some contexts a node (i.e., the entity behind that it models) wants to increase its rank.

In general, rank improvement strongly depends on the metric behind the ranking and usually comes with a significant cost (e.g., time spent, money, etc.) that depends on many factors, so the question of what strategy a node should use to improve its rank arises. For example, in a social network, one might think of improving the rank of a given person by establishing new relationships (links) with other people (nodes). But then, two other questions would follow: *"who is it better to establish new relationships with?"* and *"how many new relationships are needed to reach a given (target) rank?"*. The answer to these questions is not trivial and requires the formulation of a solution that involves cost optimization, which is, in some cases, computationally expensive.

Several studies [326–331] focused on the improvement of the PageRank value of nodes when it is employed for the ranking, which is common in the trust context, by creating new incoming links.

Such problem, also known as *Link Building problem* or *Back-link problem*, has been initially formulated by Olsen in [331]. Informally, given a node x and a rank goal t, it consists in finding the minimum set S of nodes of the network such that if we establish new links from these nodes to x its rank improves to the t-th position. A brute force approach that evaluates all the possible subsets of network's nodes and selects the smaller one is computationally expensive and practically unfeasible even for small networks (as will be detailed later). In fact, it has been show in [330] that the link building problem is NP–hard.

To tackle this problem, several heuristics that exploit domain knowledge to produce sub-optimal but acceptable solutions have been proposed. However, there is still plenty of room for improvement as most of them are far from optimal and still are computationally expensive. We refer the Reader to [326] for a survey.

In this work, we present a novel approach named *LB–GDM* (*Link Building solution based on GDM*) that is based on Machine Learning — specifically on Geometric Deep Learning, the new research area that bridges Deep Learning and non-Euclidean data [332] — and that is inspired to the *GDM* framework, proposed in [1] to solve the network dismantling problem and discussed in Chapter 3. In particular, we employ the *Graph Attention Network* [164] layers, whose capabilities have been extensively demonstrated in the literature [333]. One of

the most important advantages of our method is that it scales well even to large networks thanks to its very low computational complexity.

The contribution of this work is not limited to a novel solution for the link building problem, as the proposed methodology can be easily extended to other similar problems, such as those involving other centrality metrics.

To validate our proposal, we tested it to 19 real-world networks and analyzed the results in terms of performance, finding that our approach performs significantly better than the state-of-the-art heuristics while its computational complexity is comparable or even lower.

The following sections are organized as in the following. Section 4.2 discusses the state-of-the-art concerning the link building problem, while Section 4.3 formally introduces the problem. We describe our proposed method, how it works and its computational complexity in Section 4.4, whereas Section 4.5 illustrates the real-world networks used as test sets, the experiments carried out with such datasets, and the results.

4.2 Related Works

The basic idea of in-linking dates back to the PageRank algorithm [57], introduced in 1998 by Google's founders Larry Page and Sergey Brin to assess the relevance P of a web page by exploiting links towards it coming from other pages, and still inspiring Google search engine [58]. Google itself describes the algorithm in [334] as follows: *PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites*.

Many studies on PageRank exist, dealing with its computational issues or limitations [335–339] or aiming at introducing variations (optimizations or extensions) to the original formula [340–342, 327]. PageRank is still an important metrics that is currently applied in various areas related to document search, such as:

- In web search, to display pages that are relevant to a query issued by users [343];
- In the implementation of the *Who to follow* service in Twitter [344], based on shared interests and common connections;
- In E-learning scenarios, to select useful resources in specific topics [345, 346];
- In cars and humans traffic prediction, through public streets and places ranking [347];
- In a recommendation network, to suggest reliable entities to interact with [348, 349].

In [350], authors discuss other several domains — from chemistry to sports, literature, neuroscience and others — where PageRank is employed.

Avrachenkov and Litvak [351] study the effect of PageRank when a given page establishes one or more links to other pages and propose a strategy for an optimal linking acquisition. They also discuss the impact of back-links modification in some real application such as the Google ranking of Web pages.

Olsen and other [331, 352] formalize the problem of link building process, study its complexity and discuss the impact of the topology of the graph on the choice of potential new back-links. Specifically they find that link building is W[1]-hard. Furthermore, they show that this problem is in the class of NP optimization problems that allow polynomial-time approximation algorithms with approximation ratio bounded by a constant. In [330] the authors highlight that the problem is NP-hard if k is part of the input and present some algorithms for simple cases.

In a preliminary work [353], we introduce the problem for a new node to achieve the highest rank possible first establishing the desired rank value and assigning a budget to limit the cost while increasing the rank. The process involves two stages: first the node establishes a certain number of trust links with other existing nodes in order to get trusted (therefore improving its rank), then it tries to increase the rank keeping the cost below the budget as low as possible. However, a couple of questions arise:

- Which new nodes should it connect to?
- Should it preserve existing trust links or not?
- What are costs associated with these actions?

In [353, 354] some heuristics to solve back-link problem are discussed and a solution whose complexity is $O(|V| \cdot \log |V|)$ is proposed (where $|V|$ is the number of nodes in a network). Furthermore, the paper presents some results from simulation on random and scale-free networks which highlight that better rank improvement comes by acquiring long distance in-links whilst human intuition would suggest selecting neighbors.

In [327] several heuristics to address the back-link are proposed and compared performing simulations on Scale-Free and Erdős-Rényi networks. The results underline that the long-distance link approach achieves the best trade-off between cost and increase of rank.

4.3 Background and formulation

In this section we first recall the PageRank formulation, then we formalize the link-building problem, and finally we briefly describe some heuristics and their computational complexity.

4.3.1 PageRank

In this section we outline the basics of PageRank algorithm [57].

In general, PageRank can be considered a centrality measure used to score entities in a network. It assumes that a network is described as a graph [338, 355, 356] where the nodes model entities (e.g., agents, people, devices, resources and so on) and the directed links represent relationships between nodes (e.g., trustworthiness in a social network, or hyperlink in the web). To assign a score to each network's node, the PageRank employs a *random walker* that represents a web surfer. The surfer moves from a page to another by selecting one of its out-going links randomly: each link of a web page (the node) has the same probability $1/k^{out}$ of being followed, where k^{out} is the *out-degree* (the number of out-going links of the node). That is, the probability distribution is uniform. However, to cope with the problem of the so-called *sink nodes* [57] — i.e. those nodes with no out-going links that would trap the random walker — at each step the random walker also has a chance of jumping to a random node (that is, it does not follow any out-link).

To formalize such behavior, let us consider a network represented as a graph $\mathscr{G} = (V,E)$ where V is the set of nodes (or vertices) and E are the directed links (edges) among nodes. The number of nodes in the graph is $n = |V|$ while the number of links is $e = |E|$. A link e_{ij} from a node i to a node j is represented by an ordered couple $(i, j) \in E$.

A common and useful way to represent a graph is through its *adjacency matrix* A (of $n \times n$ size), where each A_{ij} is 1 if there is a link going from node i to node j and 0 otherwise.

We also identify with $\mathscr{N}_i^{in}$ the in–neighborhood of the node i, i.e. the set of nodes $\mathscr{N}_i^{in} = \{j \in V : \exists(j,i) \in E\}$, and with $\mathscr{N}_i^{out}$ the out–neighborhood of i, i.e. the set of nodes $\mathscr{N}_i^{out} = \{j \in V : \exists(i,j) \in E\}$. In the following we indicate with k_i^{in} and k_i^{out} respectively the input and out degree of a generic node i, i.e. the number of links incoming/outgoing to/from i.

Formally:

$$k_i^{in} = \sum_{j \in \mathscr{N}_i^{in}} A_{ji} \qquad k_i^{out} = \sum_{j \in \mathscr{N}_i^{out}} A_{ij} \tag{4.1}$$

Let us introduce the link matrix L $(n \times n)$ defined as:

$$L_{ij} = \begin{cases} 1/k_i^{out} & \text{if } A_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

The *sink vector* H $(n \times 1)$ is defined as:

$$H_i = \begin{cases} 1 & \text{if } k_i^{out} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

K is the *personalization vector* of size $1 \times n$. While this vector can be arbitrarily chosen as long as it is stochastic, a common choice is to make its values all equal to N^{-1}. $T = \mathbf{1}_{n\times 1}$ is the *teleportation vector*, where each element is 1.

As described in [57], the *transition matrix* M, used in the associated random walker problem, is derived from the link matrix, the sinks vector, the teleportation vector and the personalization vector defined above:

$$M = d(L + HK) + (1-d)\,TK \tag{4.4}$$

where $d \in [0,1]$ is called *damping factor* and, in the original implementation [57], it is set to 0.85.

The random surfer model of the PageRank can be mapped to a Markov chain in which the states are the networks' nodes and the transitions are the links between nodes. As we know from the Markov chain theory, the random walk probability vector at step n can be calculated as:

$$P_n = M^{\top} P_{n-1} \tag{4.5}$$

the related random walker problem can be calculated as:

$$P = \left(\lim_{n\to\infty} M^n\right)^{\top} P_0 = \lim_{n\to\infty} (M^{\top})^n P_0 = M_{\infty}^{\top} P_0 \tag{4.6}$$

Each element $P^{(i)}$ of the probability vector is the PageRank value of a node i, whose interpretation is *'the probability for the random surfer of being at a node at any given point in time during the walk'*.

The semantics is that the PageRank algorithm will assign high PageRank values to nodes that would appear more often in a random surfer walk, i.e. nodes with high PageRank are

relevant nodes that will be more visited by surfers and this ranking came from the link structure of the graph.

Finally, in the following we assign the nodes a rank according to the decreasing value of PageRank, so that the node with the highest value of PageRank in graph $\mathscr{G}$ is the first in the rank. We indicate with $R_{\mathscr{G}}^{(i)}$ the position of the node i in such a list (e.g. $R_{\mathscr{G}}^{(i)} = 1$ if node i has the highest value of PageRank in the graph $\mathscr{G}$).

4.3.2 The link building problem

The PageRank value of a node in a network depends on the nodes it is connected to. This means that a node x joining the network $\mathscr{G}$ can firstly choose its rank t by appropriately selecting the set of nodes in the network to be pointed by (i.e. its in-neighborhood $\mathscr{N}_x^{in}$). As discussed in the previous sections, each link comes with a cost and ideally the in-neighborhood of x should be the minimum necessary to reach the rank t in order to keep the cost as low as possible.

The *link building* (or *best in–attachment*) problem can be formulated as the problem of finding the minimum set of nodes S — also called optimal in-attachment set — such that the $R_{\mathscr{G}'}^{(x)} = t$ if we connect nodes in S to x. $\mathscr{G}' = (V \cup \{x\}, E \cup (S \times \{x\}))$ is the graph obtained by adding the set of $s = |S|$ directed links from nodes in S to the node x.

Essentially, we establish s new links from the nodes in S in order to have x's rank value equal to t. More formally S is the minimum set such that:

$$S \subseteq V : R_{\mathscr{G}'}^{(x)} = t \tag{4.7}$$

The link building is an NP-hard problem [329], and, in general, it is computationally challenging to explore all the possible combinations of nodes in order to find the minimum set S. In fact, to compute the best rank of x with s nodes, we need to explore $\binom{n}{s}$ combinations. Note that for each combination we need: 1) to compute the PageRank values of all network's nodes and 2) to sort nodes according to the decreasing value of PageRank. Moreover, given a target rank t, we don't know a-priori the minimum number of in-neighborhood nodes (i.e. the cardinality s of S) so that x rank $R_{\mathscr{G}'}^{(x)} = t$. For this reason it may be necessary to explore 2^n combinations in the worst case, which makes the computation of the optimal solution in real-life scenarios unfeasible. For instance, a network with only 100 nodes needs more than 10^{30} computations of the PageRank in the worst case. Due to these computational issues, it is necessary to find an approximation algorithm to choose a solution acceptably close to the optimum in a polynomial time.

4.3.3 State-of-the-art heuristics

In this sub-section we provide an overview of the state-of-the-art heuristics in the literature [357, 358, 327, 353]. We also report the computational complexity of these heuristics, which often depends on the number of attachments s needed to reach the desired target. For a more in-depth analysis, we refer the Reader to the original works referenced.

The link-building heuristics can be divided in problem-agnostic and problem-aware strategies.

4.3.3.1 Problem-agnostic strategies

Heuristics that do not exploit any problem-specific knowledge belong to this category, for instance:

- *Random.* Randomly choose the nodes to get in-links from. This approach is, of course, computationally inexpensive and its computational complexity only depends on the number of steps s needed to converge, i.e., it is ($\mathcal{O}(s)$).
- *Degree based.* Nodes are chosen according to their in-degree (*In Degree*) or out-degree (*Out Degree*), or even a combination of the two. This family of strategies is $\mathcal{O}(|E|)$, where $|E|$ is the number of edges in the network.
- *Long-distance based.* Nodes farther from the newcomer x are chosen first [327]. If classic Dijkstra's algorithm with Fibonacci heap is used, this approach is:

$$\mathcal{O}(|V| \cdot log(|V|) \; + \; |E|)$$

  where $|V|$ is the number of nodes in the network.

4.3.3.2 Problem-aware strategies

These heuristics exploit specific information, e.g., focusing on the node metric used for ranking; while these strategies tend to achieve better results, they usually have higher computational complexity. For instance, the classical link building problem using PageRank has a final computational complexity with a multiplicative factor of $\mathcal{O}(PR)$, where:

$$\mathcal{O}(\text{PR}) = \mathcal{O}(|V|^3)$$

if the exact Gauss method is used, or, using the approximated power method:

$$\mathcal{O}(\text{PR}) = m\,|E|$$

where m is the number of iterations needed to get a good approximation.

The problem-aware strategies are introduced in the following.

- *Anticipated Value.* This static strategy computes the value of the PageRank used for the ranking at the beginning of the link-building process and picks nodes with higher value first. The PageRank values are not recomputed during the process. Its computational complexity is:

$$\mathcal{O}(\text{AnticipatedValue}) = \mathcal{O}(\text{PR})$$

- *Current Rank.* Dynamic version of the *Anticipated Value* heuristic. That is, predictions are computed before any attachment. The computational complexity is:

$$\mathcal{O}(\text{CurrentRank}) = \mathcal{O}(s) \cdot \mathcal{O}(\text{AnticipatedValue})$$

- *Anticipated Out-degree.* Similar to the *Anticipated Value* but the ratio between the node centrality metric and the out-degree is used. This heuristic is tailored for the PageRank algorithm as nodes with higher rank value and lower degree are supposed to transfer most of the centrality value. Its computational complexity is:

$$\mathcal{O}(\text{AnticipatedOutDegree}) = \mathcal{O}(s) \cdot O(\text{PR})$$

- *Future Rank.* This strategy moves across local maxima by picking at each attachment step the node that provides the target node x with the maximum PageRank value. The computational complexity of this algorithm is:

$$\mathcal{O}(\text{FutureRank}) = \mathcal{O}(s) \cdot O(|V|) \cdot \mathcal{O}(\text{PR})$$

  which makes this heuristic feasible for small networks only.

Table 4.1 summarizes the previously discussed heuristics and their complexity.

Strategy	Name	Complexity
Problem-agnostic	Random	$\mathscr{O}(s)$
	Degree based (In- & Out-)	$\mathscr{O}(\|E\|)$
	Long distance	$\mathscr{O}(\|V\| \cdot log(\|V\|) + \|E\|)$
Problem-aware	Anticipated Value	$\mathscr{O}(PR)$
	Current Rank	$\mathscr{O}(s) \cdot \mathscr{O}(\mathrm{PR})$
	Anticipated Out-degree	$\mathscr{O}(s) \cdot O(\mathrm{PR})$
	Future Rank	$\mathscr{O}(s) \cdot O(\|V\|) \cdot \mathscr{O}(\mathrm{PR})$

Table 4.1 Summary of heuristics and their computational complexity.

4.4 LB–GDM

In this section we describe our method, how it works, its computational complexity, the model architecture, the node features used, and the training data.

4.4.1 Model architecture and complexity

In this work we adopt the same architecture as the one proposed for GDM [1] in Chapter 3. Specifically, the proposed architecture, summarized in Fig. 4.1a, is a stack of a variable number of graph convolutional layers — specifically *Graph Attention Network* (GAT) [164] layers — and a Multi-Layer Perceptron (MLP) that performs regression and outputs the prediction value p_n for each node n. Each convolutional layer is coupled with a linear layer that works as a residual connection (i.e., the input to the layer is fed to both and then outputs are sum together) and the Exponential Linear Unit (ELU) activation function is applied to the output. After the regression phase we use a sigmoid activation function so that the p_n values are in the $[0,1]$ range. The model parameters can be found in Section 4.6.

As already discussed for GDM in Section 3.2.1, the choice of the Graph Attention Network layers comes with a number of advantages:

- They process the whole neighborhood of each node, effectively aggregating the local information using the so-called *heads*, while many graph convolutional layers perform sampling;
- They learn and assign a relative-importance value to each neighboring node thanks to the attention mechanism borrowed from Natural Language Processing area and used to scale the features incoming from that node;

- Like the multiple filters in the classical convolution network, the aggregation phase can be performed multiple times and the results are concatenated or sum together;
- They scale well, considering their low computational complexity — $\mathcal{O}(\text{GAT}) = \mathcal{O}(h(|V| + |E|))$ where $|V|$ and $|E|$ are the number of nodes and links in the network and h is the number of *heads* —, because the number of operations carried out by the convolution operator depends on the number of links in the network.

Regarding the node features, we employ computationally cheap node features for each node as we extend the ones used in GDM [1] to the case of directed networks, and do not recompute the features or the predictions during the link-building process. Specifically, we use the in- and out- degree, K–Core and local-clustering coefficient. These features are both local (the degrees), second-order (the clustering coefficient) and global (the k-coreness), and provide useful information when propagated to the L-hops neighbors. More in detail, for each node $i \in V$, we compute each feature as follows:

- The in- and out- degree of i, k_i^{in} and k_i^{out}, are computed as in the equation 4.1;
- The local-cluster coefficient of i is computed by using the definition given in [22]:

$$C_i = \frac{\text{number of triangles connected to vertex } i}{\text{number of triples centered on vertex } i}$$

  informally it quantifies how close is the neighborhood of i to being a clique (i.e. a complete sub-graph);
- Given a graph $\mathcal{G}$, a k–core is a maximal connected sub-graph of $\mathcal{G}$ in which all vertices have degree at least k [359], the k–coreness of a vertex i is k if it belongs to a k–core but not to a (k+1)–core.

The computational complexity of our approach depends on the graph-convolutional network layers used and on the input features computed during the pre-processing phase:

$$\mathcal{O}(\text{LB-GDM}) \approx max\{\mathcal{O}(\text{GAT}),\ \mathcal{O}(\text{Features})\}$$

The node features are:

$$\mathcal{O}(\text{Features}) \approx \max\{\mathcal{O}(\text{Features})\} = \mathcal{O}(\text{K}-\text{Core}) \tag{4.8}$$

where

$$\mathcal{O}(K\text{-}Core) = \mathcal{O}(|V| + |E|) \tag{4.9}$$

Summing up, the total time complexity of LB–GDM is:

$$\mathscr{O}(\text{LB-GDM}) \approx \mathscr{O}(h(|V|+|E|))$$

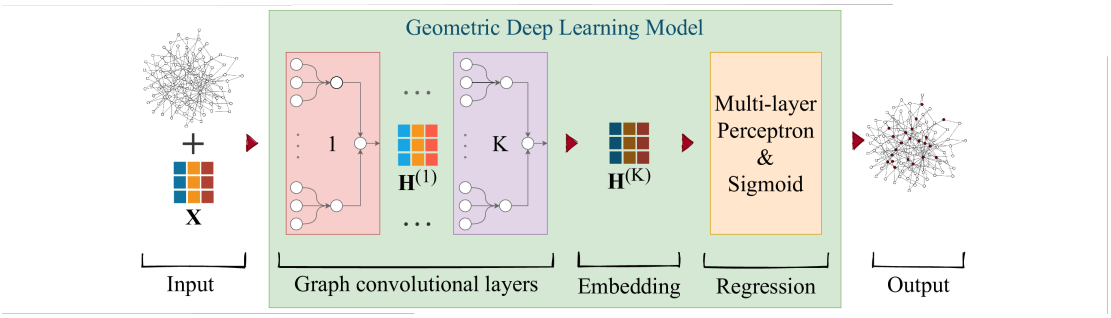
which can be approximated to $\mathscr{O}(|E|)$ or to $\mathscr{O}(|V|)$ for sparse networks, considering the low number of heads h used.

4.4.2 Training and Generalization

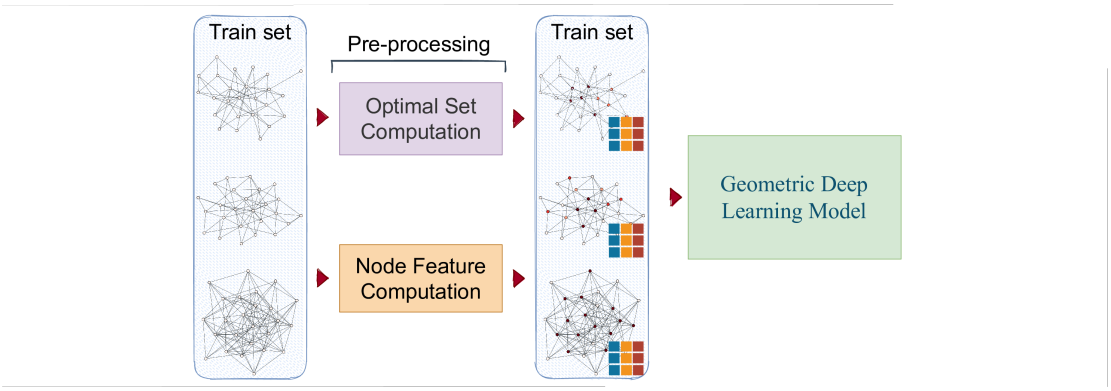
The graph convolutional network layers show remarkable inductive capabilities, can learn from very small networks (e.g. tens of nodes) and generalize to way larger ones (even millions of nodes) [146]. For this reason, we build the training set from small synthetic networks generated using the Static Power law model. More in detail, we generate 100 networks of 20 nodes and 100 links each, with different in- and out- degree distribution combinations ($2.0 \leq \gamma^{in}, \gamma^{out} \leq 3.0$) [360]. In Fig. 4.1b the train set generation and the training phases are shown.

In particular, nodes in the train networks are assigned a learning target value that depends on their belonging to any optimal set S for that network, i.e., to the sets of minimum cardinality that grant the target rank $t = 1$. Specifically, each node i of the train networks has a label value $y_i = S_i/S_t$, where S_t and S_i are, respectively, the total number of optimal sets and the number of those sets i belongs to. That is, $y_i \in [0,1]$. This training target aims at teaching the model what nodes are more relevant to reach the top rank position.

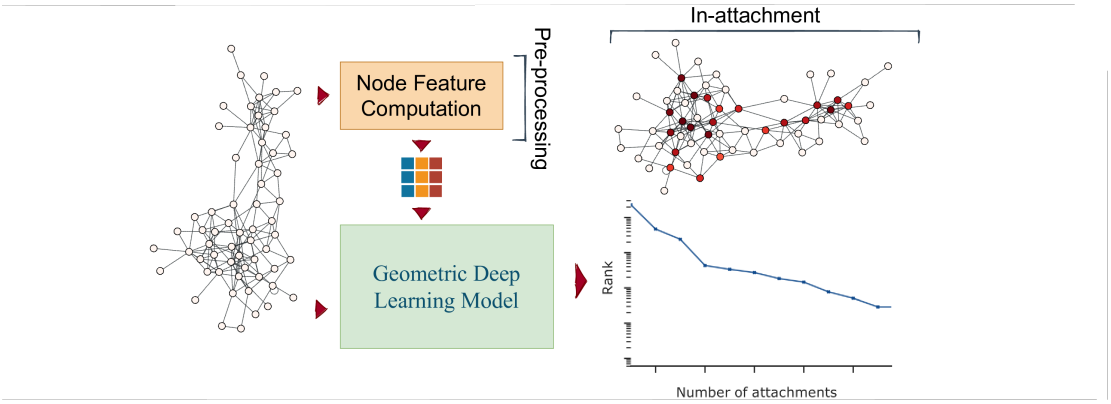
After training the model, it can generalize to new networks providing a score for each node. In detail, given a network, the link-building process with our approach, depicted in Fig. 4.1c, consists of getting the node predictions p_i (i.e., computing the node features and feeding the network plus the features to the model), sorting the nodes in descending p_i order and creating a link from the nodes on the top to the target node x until the in-attachment target rank is reached.



(a) **LB-GDM's model architecture.** Our Geometric Deep Learning model architecture as K graph convolutional layers (specifically, Graph Attention Network, GAT) coupled with a linear layer each — used as residual connections, not shown in the figure for sake of simplicity — followed by a Multi-layer Perceptron (MLP) and a sigmoid activation function that perform regression on the nodes and return, for each, a value between 0 and 1. The model takes as input the network and its node features ($\mathbf{X}$), while $\mathbf{H}^{(k)}$ is the tensor of nodes' embedding for each convolutional layer k.



(b) **Dataset generation and training phase.** The models are trained on small synthetic networks, where exploring all the solutions to find the optimal one(s) is feasible in reasonable time. In particular, for the training target, we find the optimal solutions — i.e., the smallest S set(s) — that provide the first position to a newcomer node, and assign to each node i a score p_i equal to the ratio of optimal solutions i belongs to.



(c) **Generalization to new networks.** After training the model, it can generalize to (previously unseen) networks and provide a score for each node, used to rank them and build the attachment set S.

Fig. 4.1 **Overview of our approach.** The proposed approach consists into two phases: dataset generation and training phase, and the generalization phase.

Network	Name	Category	$\|V\|$	$\|E\|$	References
advogato	Advogato trust network	Social	5K	47.1K	[223, 224]
as-caida20040105	Internet AS	Computer	16.3K	65.9K	
cfinder-google	Google.com internal	Hyperlink	15.8K	170.3K	[227, 228]
cit-HepPh	arXiv hep-ph	Citation	34.4K	421.4K	[361, 247]
cit-HepTh	arXiv hep-th	Citation	27.4K	352.5K	[362, 247]
dblp-cite	DBLP	Citation	12.5K	49.7K	[233, 234]
ego-gplus	Google+ (NIPS)	Social	23.6K	39.2K	[363, 245]
foldoc	FOLDOC	Hyperlink	13.4K	120.2K	[364, 365]
friendfeed	FriendFeed	Social	5.5K	31.9K	[366, 367]
moreno_blogs	Political blogs	Hyperlink	1.2K	19K	[368, 240]
moreno_health	Adolescent friendships	Friendship	2.5K	13K	[369, 370]
openflights	OpenFlights (Patokallio)	Infrastructure	3.4K	37.5K	[371]
p2p-Gnutella31	Gnutella hosts (31 Aug 2002)	Computer	62.6K	147.9K	[294, 292]
physician_friend	Physicians trust	Trust	110	228	[372, 373]
soc-Epinions1	Epinions	Trust	75.9K	508.8K	[374, 375]
soc-sign-bitcoinotc	Bitcoin OTC	Trust	5.9K	35.6K	[376, 377]
subelj_cora	Cora	Citation	23.2K	91.5K	[378, 379]
twitter	Twitter	Social	5.6K	42.3K	[366, 367]
wiki-Vote	Wikipedia elections	Vote	7.1K	103.7K	[380, 381]

Table 4.2 **Real-world test networks table.**

4.5 Experiments

4.5.1 Test networks

As mentioned in the introduction, we test our proposal on 19 real-world directed networks covering a wide spectrum of domains as social, hyperlink, citation, vote and technological networks with up to $76K$ nodes and $510K$ edges. We provide detailed information about the networks as name, category, number of nodes ($|V|$), edges ($|E|$), and references in Table 4.2.

4.5.2 Results

To evaluate the performance of our approach to the link-building problem, we choose the *Area Under the Curve* (AUC) as it accounts for how well an heuristic performs during the entire process, and allows for an extensive comparison on large test sets. For each heuristic and network, we calculate the AUC value using Simpson's rule on the $y(l) = R_G^{(i)}(l)/|V|$ curve, where $R_G^{(i)}(l)$ is node i's rank in function of the number of in-links created l; the lower AUC the better (average) ranking during the in-attachment.

We test our approach on the datasets described above and stop the in-attachment when the newcomer node x reaches the first position. The cumulative results, i.e., the sum of

the AUC values of all the networks (the lower, the better), for each heuristic are shown in Fig. 4.2 in increasing cumulative AUC order, and a subset of the attachment curves is shown in Fig. 4.3. We also report the full results in tabular form in Table 4.3, where, in order to facilitate the comparison, values must be interpreted as the percentage of the AUC value scored by LB–GDM for the same network (i.e., values greater than 100 mean that LB–GDM outperforms the heuristic and vice-versa).

Our results show that not only LB–GDM performs significantly better than the state-of-the-art heuristics, reaching the first position with fewer links (smaller S) and achieving better rank during the whole process, but also that its computational complexity is comparable or even lower, as discussed in the previous sections.

According to these results, the closer heuristics are *Anticipated Value* and its dynamic version *Current Rank*. Both score an AUC value 80% higher while requiring the computation of the PageRank.

Regarding the *Future Rank* heuristic, its very high computational complexity allows only evaluation on a subset of smaller test networks where it finishes in a reasonable time.

The comparison, available in Table 4.3, shows that LB–GDM is able to match its performance or provide similar AUC values in many networks, and that, on average, its AUC is just 7.0% higher, which is remarkable considering its extremely lower computational time complexity.

The implementation of our models relies on PyTorch Geometric library [161] on-top of PyTorch [382]. The graph data structures and their plots, the link creation algorithms and the PageRank computation are implemented using the graph-tool [213] library.

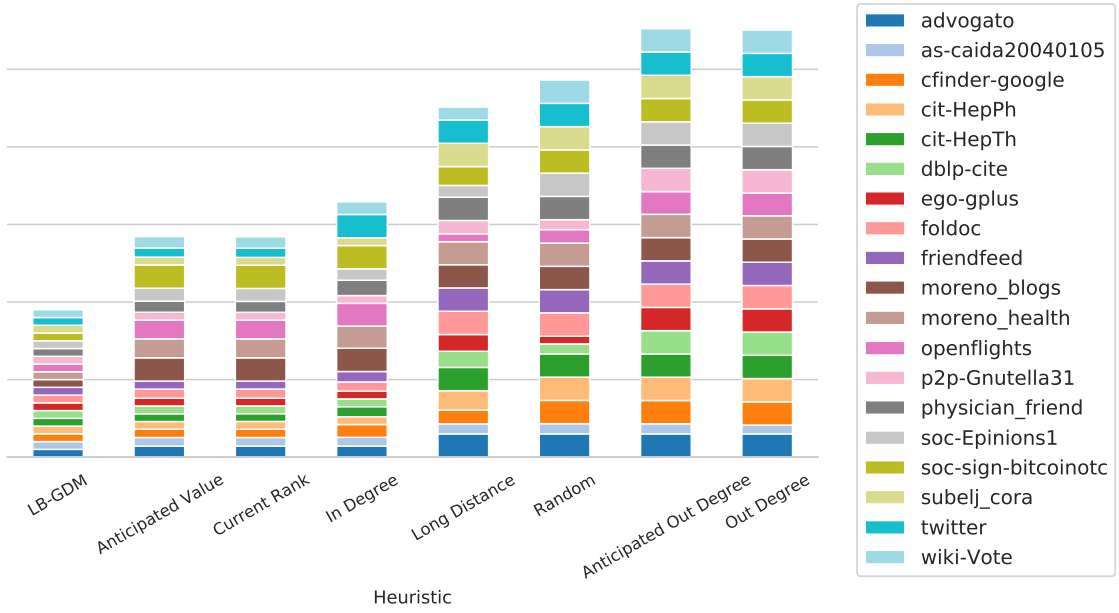
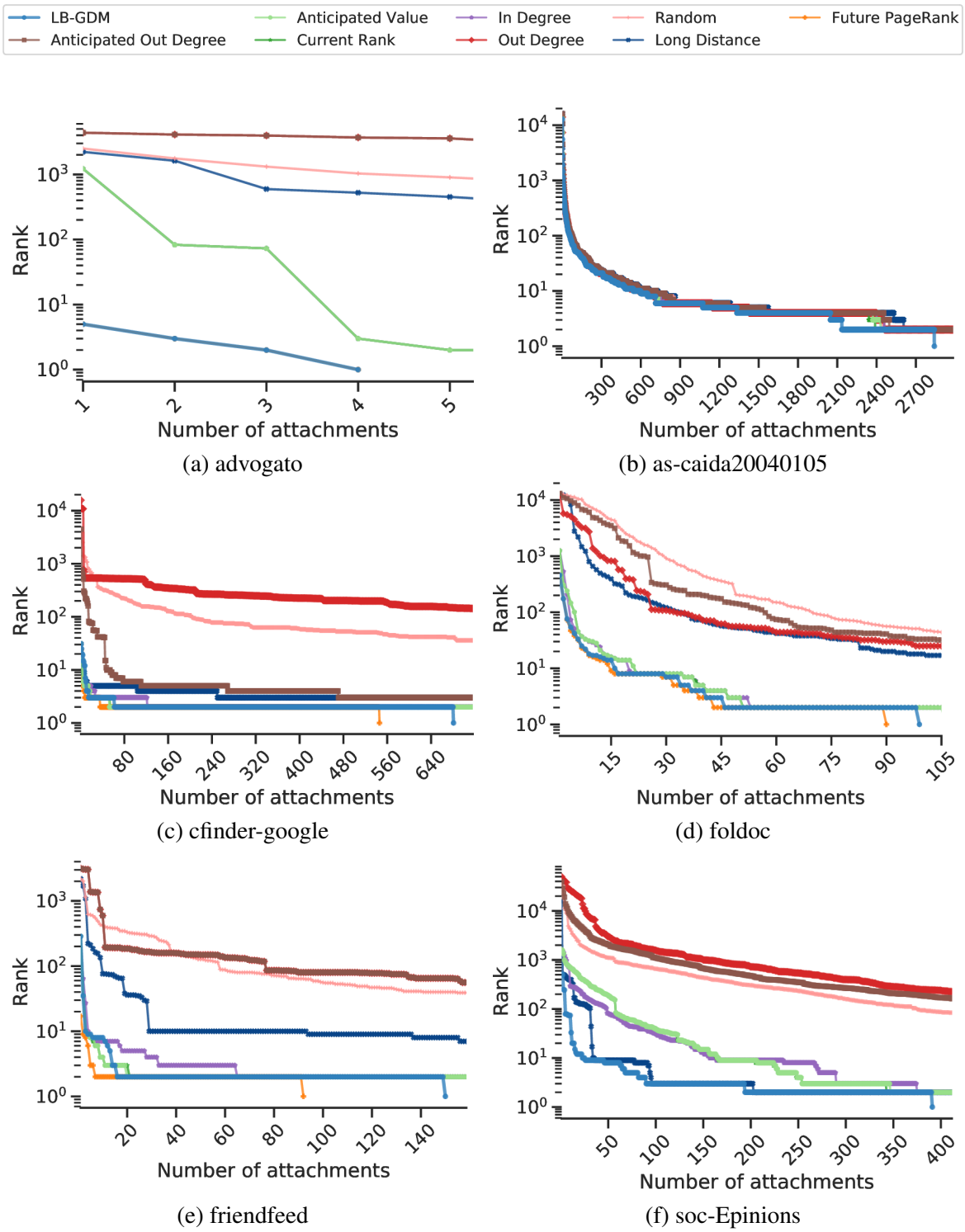


Fig. 4.2 **Link-building in real-world networks.** Per-method cumulative area under the curve (AUC) of link-building in real-world networks. The lower, the better. The target rank is the first position. Each value is scaled to the one of our approach (LB–GDM) for the same network. Note that some values are clipped to 3x to improve visualization.

Heuristic Network	LB–GDM	Future PageRank	Anticipated Value	Current Rank	In Degree	Long Distance	Random	Anticipated Out Degree	Out Degree
advogato	100.0	100.0	141.9	142.0	142.2	306.0	626.5	3438.1	3438.1
as-caida20040105	100.0	-	110.9	110.6	112.7	129.2	131.3	130.4	112.7
cfinder-google	100.0	95.8	106.3	107.5	165.6	179.5	1466.0	369.7	4079.7
cit-HepPh	100.0	99.9	99.9	99.9	100.5	248.0	498.2	1869.2	15981.1
cit-HepTh	100.0	99.4	100.3	100.2	129.9	326.8	1075.6	2530.0	13451.3
dblp-cite	100.0	100.0	100.1	100.1	100.0	208.2	128.7	712.9	2757.2
ego-gplus	100.0	100.0	100.0	100.0	100.0	216.1	100.0	1120.6	1121.4
foldoc	100.0	99.2	117.5	117.6	118.6	889.7	2377.7	1756.2	884.5
friendfeed	100.0	86.5	103.7	102.7	134.4	356.9	1150.0	1687.8	1688.3
moreno_blogs	100.0	99.5	344.6	352.7	309.0	307.3	1480.1	5188.2	4934.4
moreno_health	100.0	99.6	244.5	244.0	285.6	297.1	998.4	1140.8	1140.8
openflights	100.0	43.2	241.9	241.6	293.1	102.5	173.6	293.4	293.4
p2p-Gnutella31	100.0	100.0	100.8	100.8	100.6	170.7	127.9	473.9	2600.8
physician_friend	100.0	100.0	143.7	141.1	198.0	415.9	539.7	617.9	584.8
soc-Epinions1	100.0	-	166.7	166.8	146.9	151.8	991.8	1502.4	2864.0
soc-sign-bitcoinotc	100.0	77.1	835.3	835.6	808.2	243.5	718.2	1162.7	1152.4
subelj_cora	100.0	99.9	100.4	100.4	100.7	428.0	1011.9	676.6	5814.9
twitter	100.0	87.0	117.4	118.0	401.2	379.1	768.6	3455.7	3455.7
wiki-Vote	100.0	99.9	147.1	147.1	162.8	166.1	443.3	9011.9	8925.8
Average	100.0	93.4	180.2	180.5	205.8	290.6	779.4	1954.6	3962.2

Table 4.3 **Full results table.** To improve readability, for each network and method, we report the result as the percentage of the value scored by LB–GDM for the same network. That is, if the value is greater than 100 LB–GDM outperforms the method, and is outperformed otherwise. Regarding the Future PageRank heuristic, two networks are omitted as the heuristic would not complete in reasonable time (i.e., less than a week on our server-grade hardware).



(a) advogato

(b) as-caida20040105

(c) cfinder-google

(d) foldoc

(e) friendfeed

(f) soc-Epinions

Fig. 4.3 **Link-building in real-world networks.** Attachment curves of the networks in our test set. The y-axis value is the rank of the target node as a function of the number of in-links added. LB–GDM performs better than the cutting-edge heuristics as not only it provides better AUC (i.e., the rank is — on average — lower with the same number of new links), but also requires fewer links to reach the target in many cases.

4.6 Model parameters

We perform a grid search to test various combination of model parameters, reported here, and select the models that better fit the attachment target.

- Convolutional layers: *Graph Attention Networks* (GAT):
  - Number of layers: from 1 to 4;
  - Output channels for each layer: 5, 10, 20, 30, 40 or 50, sometimes with a decreasing value between consecutive layers;
  - Multi-head attentions: 1, 5, 10, 15, 20 or 30 with concatenation. Last layer's heads are sum together;
  - Dropout probability: fixed to 0.3;
  - Leaky ReLU angle of the negative slope: fixed to 0.2;
  - Each layer learns an additive bias;
  - Each layer is coupled with a linear layer with the same number of input and output channels in order to create residual connections from one layer to the one below;
  - Activation function: Exponential Linear Unit (ELU). The input at each convolutional layer is the sum between the output of the GAT and the linear layers;
- Regressor: Multi Layer Perceptron:
  - Number of layers: from 1 to 4;
  - Number of neurons per layer: 20, 30, 40, 50 or 100, sometimes with a decreasing value between consecutive layers.
- Learning rate: fixed to 10^{-5};
- Epochs: we train each model for 50 epochs;

4.7 Discussion

In this work we proposed a solution to the link building problem based on Geometric Deep Learning. Tests conducted on several real-world networks show that our low computational complexity approach outperforms the state-of-the-art algorithms, proving its validity. Moreover, the methodology proposed is general and can be applied to other similar problems, e.g., the ones that involve other centrality measures.

Chapter 5

mGNN: Generalizing the Graph Neural Networks to the Multilayer Case

5.1 Introduction

Multi-layer networks are pervasive in many fields, since many empirical systems exhibit multiple types of interactions or relationships simultaneously. This introduces a new level of complexity and topological correlations [84] which required the development of an *ad hoc* mathematical framework [80]. Moreover, as in the monoplex case, many problems on multi-layer networks are hard to define or computationally hard to solve. But, unlike in the case of monoplex networks, there are no Geometric Deep Learning algorithms suitable for multi-layer networks, a significant knowledge gap in the literature considering that, as already discussed in Section 1.1.6.2, aggregating the layers of multi-layer networks is not an option.

In this work we propose *mGNN* an extension of Graph Neural Networks to deal with multi-layer networks. Specifically, we propose a framework able to manage both the intra- and inter–layer relations of these networks by using any kind of GNN layers. To validate our framework, we show its application to solve three real–world problems on multi-layer networks: nodes (genes) classification in a genetic multi-layer network related to malaria, link prediction in a multiplex social network (FriendFeed, Twitter and YouTube users) and multi-layer network classification in a super-diffusion [88, 383] prediction problem.

5.2 Related Works

While, as discussed in the introduction, GNN layers have been widely used to solve many important problems, they are meant to work with monoplex networks and cannot be used to process multi-layer networks directly. Considering the importance of multi-layer networks and ubiquity, further work is needed to overcome this limitation. In fact, in the literature there are two prevalent ways to approach multi-layer networks with GNNs: 1. An aggregate-all layers approach (i.e., compressing the multi-layer network in a single graph that can be feed to a GNN). This leads to the loss of useful information of the inter-layer connections and of the different meaning/dynamics of each layer; 2. Tailoring the approach to the specific problem addressed, with the main limitation being the need of defining new methodology that cannot generalize well to other problems and that does not answer the general question of how to use GNNs in the setting of multi-layer networks.

However, some attempts in generalizing GNNs to the multi-layer case have been made, and here we analyze the most promising ones and their limitations:

- *Multi-Layered Network Embedding* (MANE) by Li et al. [384]. In their work, in order to get the multi-layer node embeddings, they optimize an objective function that is the difference of two terms: an intra-layer term, where they optimize the embeddings of nodes in each layer so that neighboring nodes are close to each other (that is, no input node feature is supported) and an inter-layer term, meant to make the embeddings of nodes in different layers close to each other if these layers depend on each other, according to the user-defined dependencies. The main limitations of this approach are: 1. trained models do not generalize to unseen networks; 2. it does not support input node features and the output embeddings are topology-based only; 3. the need of a custom weight to balance the intra- and inter- layer contribution to the objective function; 4. the need of defining layer-layer dependencies; 5. the high computational complexity ($\#iters \cdot \mathcal{O}(N^2)$, where $\#iters$ is the number of iterations needed for convergence).

- *Semi-supervised Classification in Multi-layer Graphs with Graph Convolutional Networks* (MGCN): in this work, Ghorbani et al. [385], propose a semi-supervised multi-layer node embedding framework. Specifically, they employ *Graph Convolutional Networks* (GCN) by Kipf et al. to process layers individually (i.e., a GCN per layer that does not account for inter-layer connections) and get the node embeddings, which are trained by optimizing a custom loss function composed of two parts: an unsupervised part, used to train the GCNs by reconstructing both the intra- and the inter-layer connections, and a supervised part that trains the GCNs on node classification task using node-labels for a sub-set of nodes. The authors compare against MANE, showing

better performance. The most important limitations of this approach are: 1. the need of training the GCNs to reconstruct the network, which is not useful in many tasks and creates an overhead; 2. the node embeddings are computed on each layer independently, thus the feature propagation happens only in the intra-layer; 3. it needs a custom loss function for each task and to choose a custom weight to sum the two terms of the loss.

- *Multi-GCN: Graph Convolutional Networks for Multi-View Networks, with Applications to Global Poverty* by Raza Khan et al. [386], who focus on multiplex (multi-view) networks by proposing a method to "fuse the multiple views of a graph". In particular, their approach consists of three steps: they first employ sub-space analysis to merge the layers of the network, then they identify the most informative sub-components via a manifold ranking procedure and, finally, they feed the resulting matrix to a Convolutional Neural Network (CNN) adapted to graph-structured data. The main limitations are: 1. the focus on multiplex networks; 2. the large overhead to fuse the layers and feed the result to an adapted CNN; 3. the extremely high computational complexity ($\mathcal{O}(N^3)$); 4. the small performance margin over mono-plex GNNs.

5.3 Proposed Framework

In this work we propose a novel framework to generalize existing Graph Neural Networks (GNNs) to the case of multi-layer networks. In fact, while the GNN layers have been used successfully to tackle various network science problems, extending their application to the multi-layer and multiplex networks is still an open research question.

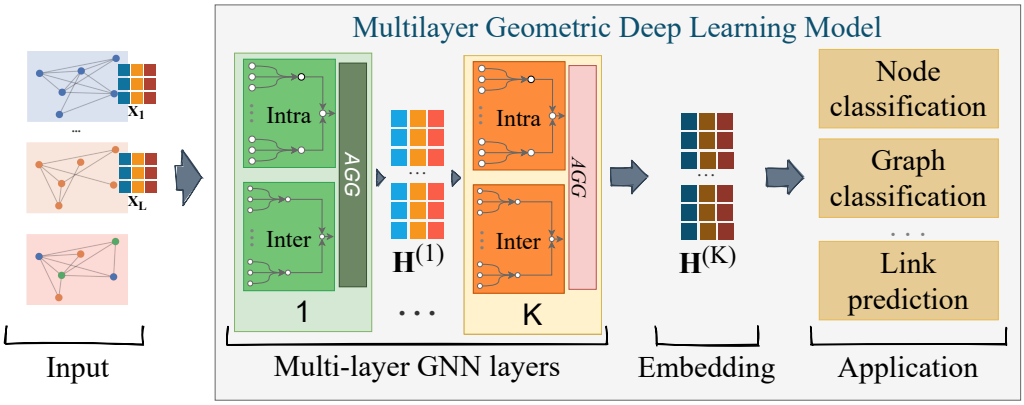


Fig. 5.1 **A Multilayer Geometric Deep Learning model example.** K is the number of convolutional layers, $\mathbf{X}_\alpha$ are the input node features of layer α. The node embeddings $\mathbf{H}^{(K)}$, can be used, like in the monoplex case, in various applications.

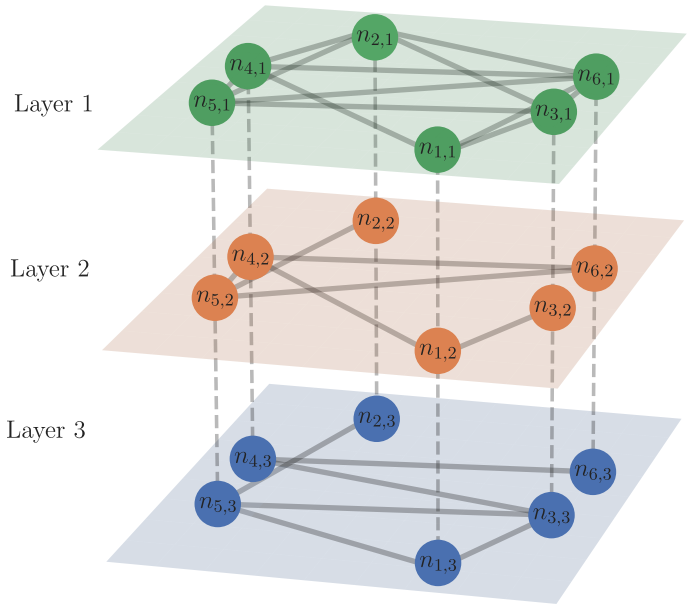
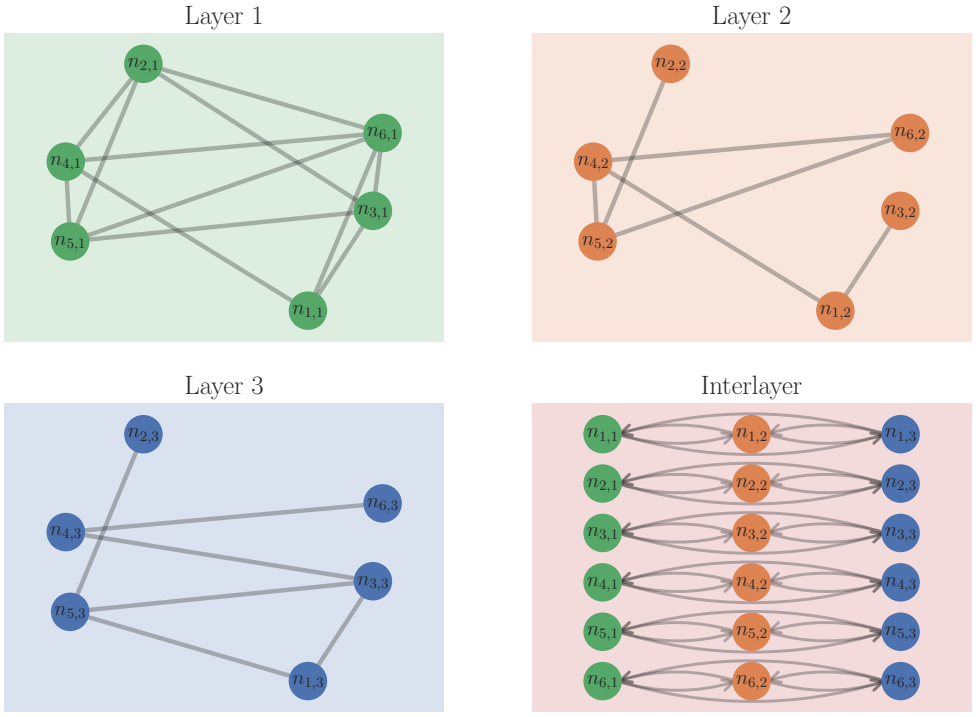
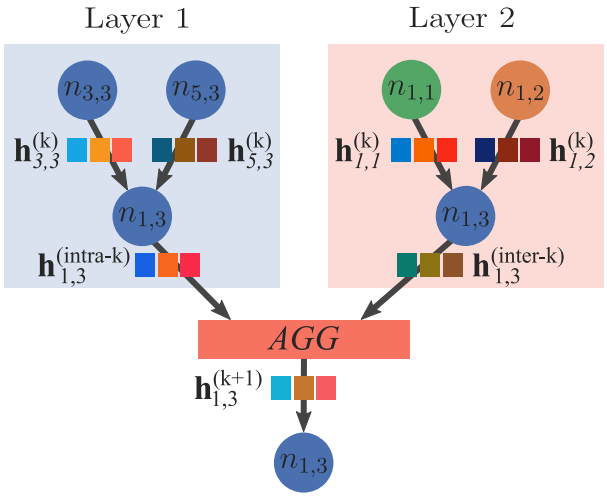


Fig. 5.2 **Example multilayer (multiplex) network.**



(a) **Step 1:** (virtually) explode the network in Fig. 5.2 into layers and interlayer links. These sub-networks will be fed to the layers along with the multilayer node features.

The framework we propose basically replaces each GNN layer with a *supra-layer* that propagates the node features both in the intra-layer and in the inter-layer neighborhoods



(b) **Step 2:** Computation of multilayer node embeddings. As an example, we show the computation of $\mathbf{h}_{11}^{(k+1)}$ (the embedding of node 1 of layer 1) after $k+1$ multiplex-convolutional layers. The intra- and inter- graph convolutional layers work as in the monoplex case (i.e., they propagate the features from the node's neighborhood), but are provided the multilayer node features as input and the output of the two is aggregated using a generic function, producing the new multilayer node embedding.

Fig. 5.3 **mGNN example.** How the multilayer convolutional layers work to produce the multilayer node embeddings.

independently — using at least two different GNN layers — and then aggregates the two output features to get the embedding of each node.

Merging the notation of the monoplex GNNs, introduced in Section 2.4.2, and the notation of the multi-layer networks, introduced in Section 1.1.6, we indicate a node i of network layer α with a tuple $i\alpha$. Like in the monoplex GNNs, and using such notation, the supra-GNN layer has the following form:

$$\mathbf{h}_{i\alpha}^{(k+1)} = AGG_{k+1}(\mathbf{h}_{i\alpha}^{(intra-k)}, \mathbf{h}_{i\alpha}^{(inter-k)}) \tag{5.1}$$

$$\mathbf{h}_{i\alpha}^{(0)} = \mathbf{x}_{i\alpha} \tag{5.2}$$

where $\mathbf{h}_{i\alpha}^{(intra-k)}$ and $\mathbf{h}_{i\alpha}^{(inter-k)}$ are computed independently using two different GNN model instances, AGG_{k+1} is some aggregation function (e.g., sum, average, a multi-layer perceptron, etc.) and $\mathbf{x}_{i\alpha}$ are the input features of the node i of layer α.

To illustrate the propagation performed by a Graph Neural Network extended to the multilayer case with our framework, we show one of the steps (performed iteratively) of node $n_{1,1}$ of the network in Fig. 5.2. Specifically, a multi-layer network with L layers is first (virtually) exploded into $L+1$ graphs — where the first L are the layers and the $L+1$-th

is the inter-layer network, which connects the nodes of different layers in the multi-layer network and depends on the network type itself —. For instance, if the network is a multiplex, the inter-layer is a network where all the replicas of a node (i.e., the $n_{i\alpha}\ \forall\alpha$) are connected in a clique (i.e., the sub-graph is fully-connected), or, if the network is a multi-layer, the inter-layer connections are the natural connections among the nodes. After exploding the network and building the (arbitrary) inter-layer graph, node $n_{1,1}$'s embedding after $k+1$ multi-convolutional layers, $\mathbf{h}_{1,1}^{(k+1)}$, are computed as the aggregation (performed by a generic and, possibly, learned *AGG* function), of its intra- and inter-layer embeddings, $\mathbf{h}_{1,1}^{(intra-k)}$ and $\mathbf{h}_{1,1}^{(inter-k)}$ respectively, which are, in turn, computed using two different GNN layers that work independently but that use $\mathbf{h}_{1,1}^{(k)}$, the multi-layer features from the layer k.

The node embeddings $\mathbf{H}^{(k)}$ produced by the supra-layers of the proposed framework (i.e., the features of nodes/replicas) can be then used as common in the monoplex case. For instance, they can be fed to a Multi-Layer Perceptron (MLP) for regression or classification, or to a pooling layer to perform layer or graph classification. As an example, if one is interested in node classification in a multiplex setting, the embedding of the replicas can be aggregated using some generic function *ReplicaAGG* as follows:

$$\mathbf{r_n} = ReplicaAGG(\{\mathbf{h}_{n\alpha}^{(k)},\ \forall\ \alpha\ \in\ L\}) \tag{5.3}$$

$\mathbf{r_n}$ is now the node embedding (i.e., it accounts for all the replicas of node n) and can be used for classifying the node.

Such framework has the advantage of decoupling the inter- and intra- layer propagation by learning two sets of GNN parameters, enabling the model to learn the different importance of the two propagation "directions", but also allowing the use of different types of GNN layers, which is useful, for instance, to provide intra-layer weights while keeping the inter-layer unweighted (or with default weights). This overcomes the major limitation of an aggregate-all-layers approach and lets the aggregation weights emerge from the training data. Moreover, there is no computational complexity overhead as it is the one of the GNN employed (i.e., $O(L\cdot GNN_{intra}+GNN_{inter})$). In comparison with other multiplex approaches, this framework: 1. is not problem-specific and can work with any multi-layer network (not just multiplexes); 2. supports any type of training (supervised, unsupervised, reinforcement); 3. does not need to train the GNNs to reconstruct the network or to define custom loss functions; 4. supports input node features and propagates them both in the intra- and in the inter- layer independently; 5. it allows stacking multiple GNN layers to capture the information in a K-hops neighborhood.

We stress that the main advantage of this framework is that already existing graph convolutional networks can be used and extended to the multi-layer case with very little effort. In addition, while we use two identical convolutional layers for each supra-layer, the configuration of the layers is arbitrary, allowing the user to choose different layer parameters or types depending on the specific domain knowledge or application (e.g., the intra-layer is weighted but the inter-layer is not). This also means that more than a single intra-GNN layer can be used. That is, each supra-layer could include up to one intra-layer GNNs per network layer, if needed to learn different parameters and capture different information or dynamics.

5.4 Experiments

To show the validity and generality of the framework proposed in this work, we test it into three different tasks:

- Node classification: classification of *var* genes of the human malaria parasite *Plasmodium falciparum*;
- Link prediction: prediction of (intra-) layer links given a multi-platform social multiplex network, where nodes correspond to users and layers to different social networks;
- Network classification: prediction of super-diffusion in multiplex networks.

We implement our framework on top of the *PyTorch Geometric* [161] library and manipulate the networks using *graph-tool* [213].

5.4.1 Malaria genes classification

In this section, we show how our framework can be used to perform node classification on a biological multiplex network. Specifically, we use the networks from the work by Larremore et al. [387], where they analyze 307 amino acid sequences from the DBLα domain of the *var* genes of seven *Plasmodium falciparum* isolates. Two nodes (genes) are connected if they exhibit a pattern of recombination, and authors find nine Highly Variable Regions (HVRs), producing an unweighted and undirected network for each. Considering that these HVRs share the same set of nodes, we build a multiplex network by using them as layers. Authors also provide classification of the sequences (nodes) into six classes, based on the number of cysteine residues present in HVR-6, and we use this information to test our framework in a classification task. Taking into account that in a multiplex network a node appears in all the layers, we reflect this relation in the inter-layer network by connecting all

the replicas of each node in a clique. Regarding the model employed, each of the supra-layers includes a *Graph Attention Network* [164] layer that processes all the intra-layers and another one for the inter-layer. The output embeddings of the replicas are transformed into the node embedding as shown in Equation 5.3, where *ReplicaAGG* is a Multi-layer Perceptron. We train in a supervised manner and test on 20% of nodes, selected using stratified sampling. That is, the test set contains the same distribution of classes as the full dataset. Considering that the dataset comes without any node feature, we do not use any as input and assign $\mathbf{x}_i = 1$ for replica node i. The final classification accuracy is 83.9%, which is remarkable considering the small size of the network, the fact that the classes are strongly unbalanced and that the model is able to classify the nodes using the topology alone without input features.

5.4.2 Link prediction

In this section, we demonstrate how models built according to our framework can be used in the multi-layer link prediction setting.

For this purpose, we build a multi-layer model with *Graph Attention Network* [164] layers (one for all the intra-layers and one for the inter-layer, which output are aggregated with a linear layer). However, compared to the previous example, we replace the node classificator with a link predictor, i.e., a Multi-Layer Perceptron that — given the embedding of two nodes — predicts the probability that the two nodes are connected. Without loss of generality, we perform intra-layer link prediction. Of course, inter-layer link prediction is still possible.

We test this model on a multiplex social network where nodes, representing users, can interact on three different social networks (FriendFeed, Twitter and YouTube) [366, 367]. More details about the *FF-TW-YT* network layers can be found in Table 5.1. In particular, we train the model to predict the existence of links in one of the layers given the full multiplex network. We perform the training phase by randomly removing 20% of links from the test layer and by feeding the remaining ones in that layer as positive examples. The removed links will be used during the test phase to evaluate the performance of the model. The negative examples are picked randomly in the same quantity among the non-existing ones, both for the train and test phases. We train and test the models without any input node feature (i.e., $x_i = 1$) on the Twitter and FriendFeed layers. The test accuracy is 83.3% on the Twitter layer (AUC score 0.829, F1 score 0.833) and 81.9% on the FriendFeed layer (AUC score 0.811, F1 score 0.819), proving that the model is effectively learning to predict the existence of links.

Layer	$\|N\|$	$\|E\|$	Type
FriendFeed	6.4K	32.0K	Directed
Twitter	6.4K	42.3K	Directed
YouTube	6.4K	0.6K	Undirected

Table 5.1 *FF-TW-YT* network layers.

5.4.3 Superdiffusion prediction

Super-diffusion is a property of certain multiplex networks where the diffusion process is faster than the diffusion on the separate layers [388], which happens if the second eigenvalue of the supra-Laplacian is greater than the maximum of the ones of the layers.

In this section we demonstrate how our framework can be used to reproduce the results from V.M. Leli et al. [389], who predict whether a multiplex network exhibits super-diffusion or not, and do so with a classical Deep Learning model employing CNNs (convolutional neural networks) on the supra-adjacency matrix. The main limitations of their work are that the models, once trained, are not able to generalize to networks with different number of nodes and layers, and that the models require a lot of training examples.

Predicting whether a multiplex network exhibits the super-diffusion property can be formulated as a network classification task. For this purpose, use the *Graph Attention Network* [164] layers (again, one for all the intra-layers and one for the inter-layer, aggregated with a linear layer) plus a *Global Soft Attention* pooling layer [160] that first transforms the node embeddings into layer embeddings (via a learned linear transformation) and then sums them to compute the network prediction.

As in the work from Leli et al., we generate the networks with two layers and 50 nodes per layer. The layers are Erdős-Rényi (ER) networks with p_i for each layer i such that

$$0 < p_1 \leq p_2 < 1$$

. We increment the p_1 and p_2 values in 0.01 steps and for each combination we generate 5 networks for the train set and 10 for the test set. As a result, we train on $\sim 5K$ networks (we balance the dataset taking all the positive examples and randomly selecting the same number of negative ones), and test on $\sim 50K$ networks. The final test accuracy is 89.2% (AUC score 0.911), with the main advantages of exploiting the graph structure and the need of way less networks required to learn and generalize.

5.5 Training and model parameters

5.5.1 Malaria genes classification

The model we use to classify the nodes has 6 supra-layers with identical sub-layers (i.e., *GAT* layers with 60 output features and 5 heads each, negative slope 0.2), followed by a Multi-Layer Perceptron with 6 outputs (the classifier) that takes as input the features of the replicas of each node and predicts its class. The model is trained for 2500 epochs (with 100 epochs patience) with learning rate $5 \cdot 10^{-4}$, weight decay 10^{-3} and 0.3 dropout probability. We use the Cross Entropy loss function with a rescaling weight to account for the different distribution of classes.

5.5.2 Link prediction

The model we use to perform link prediction has 3 identical supra-layers (i.e., *GAT* sub-layers with 30 output features and 5 heads each, negative slope 0.2). We train the model for 2500 epochs (with 400 epochs patience) with learning rate 10^{-3}, weight decay 10^{-5} and 0.3 dropout probability.

5.5.3 Superdiffusion prediction

The model we use for this task includes 4 supra-layers with identical sub-layers (i.e., *GAT* layers with 10 output features and 5 heads each, negative slope 0.2) and is trained using the Mean Square Error loss function for 100 epochs with learning rate $5 \cdot 10^{-3}$, weight decay 10^{-5} and 0.3 dropout probability. The node embeddings are aggregated using a weighted sum function learned during the training.

5.6 Discussion

In this work we present an innovative way of employing existing Graph Convolutional Networks on multi-layer networks. Compared to other works, our proposal is problem agnostic and works for any multi-layer network. Moreover, it is transparent to the training (i.e., any type of training is supported), the node feature propagation happens in both the intra- and inter- layer independently and multiple layers can be stacked to capture information from the topology and the features farther in the network.

We validate our proposal on three different tasks: multiplex node-classification, intra-layer link prediction and network classification. The results show that the approach is general

and performs well in different settings, without any computational over-head which allows the application of the method to large multi-layer networks.

Chapter 6

Weighted and Signed Graph Attention Networks

6.1 Introduction

Signed networks [41] are a class of networks, where links can be positive or negative. They are especially used to model good or bad relations among nodes. A notable example are trust networks [44–46], where nodes are users and positive/negative links among them are used to model trust/distrust relations. In general, signed networks can also be weighted, so relations can be positive or negative and with a given strength. Considering again trust networks, each link can express more or less strong relationships of trust or distrust. Weighted and signed networks are also commonly used to represent correlations networks [42, 43], where links among entities express the level of correlation that, in general, can be a positive or negative real number.

To solve many (hard) problems on networks, deep learn techniques have recently been used [145, 146, 17, 1]. In particular, Graph Neural Networks (GNNs) [159] have been employed to learn representations on graphs by abstracting from the specific application domain. GNNs are powerful tools and their applicability have been successfully demonstrated even to solve very complex problems. Among different GNN layer model, the Graph Attention Networks (GATs) [164] are one of the most promising, both in terms of performance and flexibility in solving problems in different domains. However, the original GAT formulation, discussed in Section 2.4.7.3 only takes into account undirected and unweighted networks.

In this work we propose *wsGAT*, an extension of the GAT to cope with signed and weighted networks. We show wsGAT applicability to real-world signed and weighted networks by solving the link prediction task. We compare wsGAT performance by solving

the same task with GCNII [168] and SGCN [169] models, respectively used to perform weighted and signed link prediction. Our results show that models with wsGAT layers outperform the ones with GCNII and SGCN layers.

6.2 Formulation

In this work, we extend the *Graph Attention Networks* (GAT) [164] by modifying the computation of the attention coefficient to also account for the (signed) link weight.

As common in the literature, we indicate with $\mathbf{h}_n^{(k)} \in \mathbb{R}^{F_k}$ the node embedding of node i after the k-th GNN layer, where F_k is the number of features. According to this notation, $\mathbf{h}_n^{(0)}$ are the node's input features $\mathbf{x}_n$.

In the original GAT formulation, the authors borrow the *attention* mechanism [165], defined to handle variable length sequences and used successfully in the Natural Language Processing (NLP) field, to assign a (relative) importance score to each of the neighbors of the target node. Specifically, they compute the *attention* coefficient α_{ij} of a node i for each neighboring node j as in Equation 6.1.

$$\alpha_{ij}^k = \text{softmax}(\text{LeakyReLU}(\mathbf{e}_i^k))_j \tag{6.1}$$

$$e_{ij}^k = \mathbf{a}_k^\intercal(\mathbf{W}^k\mathbf{h}_i \,\|\, \mathbf{W}^k\mathbf{h}_j) \tag{6.2}$$

where $\mathbf{W}^k \in \mathbb{R}^{F_k \times F_{k+1}}$ is a learned weight matrix, $\mathbf{a}_k^\intercal \in \mathbb{R}^{2 \cdot F_{k+1}}$ is a learned weight vector and $\|$ is the concatenation operator.

The attention coefficient is then used to scale the incoming node embedding of the neighbors as in Equation 6.3.

$$\mathbf{h}_i^{(k+1)} = f(\sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij}^k \mathbf{h}_j^{(k)}) \tag{6.3}$$

where f is an activation function, $\mathcal{N}_i$ is the neighborhood of node i, and may include the node i itself if self-loops are added to the network.

The main limitation of this formulation is that the same weight matrix $\mathbf{W}^k$ is applied independently to both of the embeddings of the target and neighboring nodes, i.e., they are combined linearly. To achieve better attention scores, other approaches that use Multi-Layer Perceptrons have been proposed [125].

We follow this trend and also account for the (signed) link weight w_{ij} in the attention computation. In detail, we first modify the computation of e_{ij}^k as follows:

$$e_{ij}^{k} = \text{MLP}^{k}(\mathbf{h}_{i}^{(k)} \,\|\, \mathbf{h}_{j}^{(k)} \,\|\, w_{ij}) \tag{6.4}$$

where MLP^{k} is a Multi-Layer Perceptron with the only requirement that the last layer can also produce negative values (e.g., a zero-centred activation function is used) and $\mathbf{w}_{ij}$ is the weight of the link.

The attention coefficients are then computed as:

$$\alpha_{ij}^{k} = \text{sign}(e_{ij}) \cdot \text{softmax}(\text{abs}(\mathbf{e}_{i}^{k}))_{j} \tag{6.5}$$

That is, in our formulation $\alpha_{ij} \in [-1, 1]$, meaning that the contribution of each neighboring node to Equation 6.3 can be positive or negative.

The choice of a Multi-Layer Perceptron allows the network to learn the relative importance of the features of the neighboring nodes j, with respect to the ones of the target node i, and is also affected by the weight and sign of the link between them.

As in the original GAT formulation, *wsGAT* also support multi-head attention, meaning that multiple embeddings for a node can be computed — each using a different set of parameters — and concatenated/sum together.

6.3 Experiments

To validate the proposed *wsGAT* layer, we test it in the link and weight prediction task on real-world trust networks.

Since, to the best of our knowledge, no other GNN layer can handle both signed and weighted links, we first decompose the final task of signed and weighted link prediction in two sub-tasks and compare our proposal against the state-of-the-art layers. Specifically, we first compare on the link sign prediction with *Signed Graph Convolutional* (SGCN), and on the (unsigned) link weight prediction with *GCNII* (Graph Convolutional Network via Initial residual and Identity mapping).

6.3.1 Dataset

We test our proposal on 4 real-world trust networks. More in detail, we test on the who-trusts-whom networks from the Advogato online community, where trust 4 trust levels can be assigned (corresponding weights are from 0.4 to 1.0 with 0.2 step), from the Bitcoin Alpha and OTC platforms, where scores are on a scale of -10 (total distrust) to +10 (total

trust), and from the Epinions.com community, where users can assign a positive or negative trust score to each other. We summarize the networks used for the experiments in Table 6.1.

Network	|V|	|E|	Positive Links	Min. Link Weight	Max. Link Weight	Refs
advogato	6,541	51,127	100%	0	1	[390]
bitcoin-alpha	3,783	24,186	89.98%	-10	10	[391]
bitcoin-otc	5,881	35,592	93.64%	-10	10	[391]
epinions	131,828	841,372	85.29%	-1	1	[392]

Table 6.1 **Dataset.** Details about the networks used in this work.

6.3.2 Sign prediction

In the first sub-task, we perform sign prediction — i.e. prediction of the kind of relationship (positive or negative) between two nodes in trust networks — and compare against *SGCN* [169].

The *SGCN* layer, to the best of our knowledge, is the only one able to handle signed links. In particular, they use balance theory and compute two feature sets for each node by splitting the node neighborhood into two sub-neighborhoods (i.e., one with all the positive links and the other with all the negative ones). That is, each node has a positive and a negative feature sets. This is a limitation from a Network Science perspective, as the two sub-networks may have different characteristics w.r.t. the original network, or disconnected components may emerge (e.g., in the case of unbalanced link signs). However, authors mitigate this issue by influencing each feature set with the other: when computing the positive node features, they also sum a function of the negative ones, and vice-versa. Another limitation of *SGCNs* is that they do not support link weights, which is useful in many contexts, like the trust one.

For a fair comparison with this approach, we use the same input spectral features and the same train methodology proposed in their paper. In detail, we provide the Signed Spectral Embedding (SSE) from [393] as input node features, and use a node classifier that predicts whether, given a pair of node embeddings, the link between the two nodes is positive, negative or non-existent. During the training phase, we provide 80% of existing links as train examples (and remove the remaining ones from the network), plus the same number of non-existing ones sampled randomly. However, by analogy with their methodology, we predict only the sign of existent links during testing.

We employ the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve and the F1-scores to evaluate the prediction performance. According to sign prediction results, shown in Table 6.2, *wsGAT* outperform the best *SGCN* algorithm on

the three signed networks in our dataset. It is worth noting that the *SGCN* results for the *epinions* network differ from the ones reported by the authors in their paper as we do not filter low-degree nodes from the graph.

GNN Layer	bitcoin-alpha		bitcoin-trust		epinions	
SGCN2	0.796	0.917	0.823	0.925	**0.842**	0.946
wsGAT	**0.832**	**0.967**	**0.845**	**0.953**	0.839	**0.949**

Table 6.2 **Sign prediction results (ROC AUC | F1).**

6.3.3 Weight prediction

In the second sub-task, we perform link weight prediction – i.e., predict the (unsigned) strength of the relationship between two nodes —. Here, we compare against *GCNII* [168] that were proposed to simplify and improve the Graph Convolutional Networks (GCN) by Kipf et al. [394]. For both *wsGAT* and *GCNII* we employ the same model architecture: after the GNN layers we use two Multi-Layer Perceptrons; while both take a pair of node embeddings as input, one is trained to predict if the existence of the link between the input nodes, the other is trained to predict the weight. Both MLPs in our tests have fixed number of layers (3) and neurons (100 neurons per layer, 1 output).

Regarding the training, we split the network links into training links (80%) and test links (20%, removed before the training). In addition, for each set we sample the same number of non-existing links to provide the negative examples, and assign a 0 weight to them.

This time we use the ROC AUC and the F1 scores to evaluate the link prediction performance, and we measure the error on the weight prediction (only for existing links) with the Mean Absolute Error (MAE). The weight prediction results are reported in Table 6.3. *wsGAT* outperform the *GCNII* not only in the link prediction task, but also predict more accurate link weights.

GNN Layer	advogato			bitcoin-alpha			bitcoin-trust		
GCNII	0.880	0.824	0.158	0.912	0.841	0.1470	0.909	0.830	0.179
wsGAT	**0.910**	**0.839**	**0.142**	**0.923**	**0.851**	**0.130**	**0.929**	**0.860**	**0.154**

Table 6.3 **Absolute weight prediction results (ROC AUC | F1 | MAE).** Note that while the higher the AUC and F1 scores the better, MAE is an error score and lower values represent smaller errors.

6.3.4 Signed weight prediction

Finally, we merge the two sub-tasks discussed previously and predict the existence of links and of their signed weight.

As in the weight prediction sub-task, we use the AUC and the F1 to measure the link prediction performance, and the MAE to measure the error on the weight prediction.

The results on the signed and weighted Bitcoin networks, reported in Table 6.4, show that the link prediction performance is almost the same as the unsigned case, and the mean absolute error (now on a scale from -10 to +10) drops significantly.

GNN Layer	bitcoin-alpha			bitcoin-trust		
wsGAT	0.922	0.839	0.069	0.921	0.852	0.079

Table 6.4 **Signed weight prediction results (ROC AUC | F1 | MAE).** Note that while the higher the AUC and F1 scores the better, MAE is an error score and lower values represent smaller errors.

6.3.5 Code availability

wsGAT were implemented on top of PyTorch Geometric [161] v1.6.3. Code will be publicly available after the publication of the paper at the following URL
https://github.com/NetworkScienceLab/wsGAT.

Chapter 7

Conclusive remarks and future research directions

The works discussed in this thesis bring two major contributions: an improvement of the state-of-the-art in two Network Science problems, and new ways to handle multi-layer and signed networks.

Specifically, we presented the GDM framework to approach various computationally hard Network Science problems and showed its application to the Network Dismantling and to the Link Building problems. In both applications, the performance of the GDM-based heuristics outperform the state-of-the-art algorithms significantly, as shown by extensive tests on empirical systems, and also have lower computational complexity. Moreover, the insights gained by explaining the models should improve the understanding of the problem and help the community. Future research directions may include extending the framework to other problems and pushing the performance by improving the datasets (e.g., by increasing the number of nodes, of networks and of generative models used), or by refining the algorithms (e.g., in the case of network dismantling, by attacking the k–core of the network).

We also presented mGNN to address the lack of GNN layers for multi-layer networks, and wsGAT, an extension of the GAT layers, to handle signed networks. Our experiments show the validity of the two proposals, and future research directions may involve the use of wsGAT layers combined via mGNN to approach problems on, e.g., brain or international-trade networks, as they usually involve multi-layer representations of the systems. It is worth mentioning that further work should be done to provide explanation tools for models built for multi-layer and temporal networks. In fact, the insights gained could help to better understand the dynamics of processes, neuroscience problems, etc.

In addition, we give a contribution to the Open Source community as we have publicly released the code of the GDM framework, along with the training networks and models, under the GPLv3 license.

References

[1] Grassia, M., De Domenico, M. & Mangioni, G. Machine learning dismantling and early-warning signals of disintegration in complex systems. *Nature Communications* **12**, 5190 (2021). URL https://doi.org/10.1038/s41467-021-25485-8.

[2] Grassia, M. & Mangioni, G. Weighted and signed graph attention networks. *Submitted to the 10th International Conference on Complex Networks and their Applications (COMPLEX NETWORKS 2021)* (2021).

[3] Grassia, M., De Domenico, M. & Mangioni, G. mGNN: Generalizing the Graph Neural Networks to the Multilayer Case. *Submitted to IEEE Transactions on Neural Networks and Learning Systems* (2021).

[4] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Efficient node rank improvement via link building using geometric deep learning. *Submitted to IEEE Transactions on Knowledge and Data Engineering* (2021).

[5] Grassia, M., Mangioni, G., Schiavo, S. & Traverso, S. International food trade and vulnerability to shocks: insights from network-based simulations. *Submitted to Environmental Research Letters* (2021).

[6] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Food recommendation in a worksite canteen. In *COMPLEXIS*, 117–124 (2021).

[7] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. A network-based analysis of a worksite canteen dataset. *Big Data and Cognitive Computing* **5**, 11 (2021).

[8] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Analysis of the co-authorship sub-networks of italian academic researchers. *Submitted to the 2021 IEEE/ACM International Conference on Advances in Social Network Analysis and Mining (ASONAM)* (2021).

[9] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Preliminary characterization of italian academic scholars by their bibliometrics. *Submitted to 14th International Symposium on Intelligent Distributed Computing (IDC 2021)* (2021).

[10] Grassia, M., Mangioni, G., Schiavo, S. & Traverso, S. (unintended) consequences of export restrictions on medical goods during the covid-19 pandemic. *arXiv preprint arXiv:2007.11941* (2020).

[11] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Group cohesion assessment in networks. In *Complex Networks XI*, 16–25 (Springer, Cham, 2020).

[12] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. A network-based analysis to understand food-habits of a multi-company canteen's customers. In *Proceedings of the 22nd International Conference on Information Integration and Web-based Applications & Services*, 352–356 (2020).

[13] Lauri, J., Dutta, S., Grassia, M. & Ajwani, D. Learning fine-grained search space pruning and heuristics for combinatorial optimization (2020). 2001.01230.

[14] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Network robustness improvement via long-range links. *Computational Social Networks* **6**, 1–16 (2019).

[15] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. A pagerank inspired approach to measure network cohesiveness. In *International Conference on Internet and Distributed Computing Systems*, 349–356 (Springer, Cham, 2019).

[16] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Strategies comparison in link building problem. In *International Symposium on Intelligent and Distributed Computing*, 197–202 (Springer, Cham, 2019).

[17] Grassia, M., Lauri, J., Dutta, S. & Ajwani, D. Learning multi-stage sparsification for maximum clique enumeration. *arXiv preprint arXiv:1910.00517* (2019).

[18] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Exploiting long distance connections to strengthen network robustness. In *International Conference on Internet and Distributed Computing Systems*, 270–277 (Springer, Cham, 2018).

[19] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Long distance in-links for ranking enhancement. In *International Symposium on Intelligent and Distributed Computing*, 3–10 (Springer, Cham, 2018).

[20] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Climbing ranking position via long-distance backlinks. In *International Conference on Internet and Distributed Computing Systems*, 100–108 (Springer, Cham, 2018).

[21] De Domenico, M. *et al.* Complexity explained (2019).

[22] Newman, M. E. J. The structure and function of complex networks. *SIAM review* **45**, 167–256 (2003).

[23] Iñiguez, G., Battiston, F. & Karsai, M. Bridging the gap between graphs and networks. *Communications Physics* **3** (2020). URL http://dx.doi.org/10.1038/s42005-020-0359-6.

[24] Brandes, U., Robins, G., McCranie, A. & Wasserman, S. What is network science? *Network Science* **1** (2013).

[25] National Research Council. *Network Science* (The National Academies Press, 2005).

[26] Boccaletti, S., Latora, V., Moreno, Y., Chavez, M. & Hwang, D.-U. Complex networks: Structure and dynamics. *Physics Reports* **424**, 175–308 (2006).

[27] da F. Costa, L. *et al.* Analyzing and modeling real-world phenomena with complex networks: A survey of applications. *Advances in Physics* **60**, 329–412 (2011).

[28] Jeong, H., Tombor, B., Albert, R., Oltvai, Z. N. & Barabási, A.-L. The large-scale organization of metabolic networks. *Nature* **407**, 651–654 (2000).

[29] Williams, R. J., Berlow, E. L., Dunne, J. A., Barabási, A.-L. & Martinez, N. D. Two degrees of separation in complex food webs. *Proceedings of the National Academy of Sciences* **99**, 12913–12916 (2002).

[30] Varela, F., Lachaux, J.-P., Rodriguez, E. & Martinerie, J. The brainweb: phase synchronization and large-scale integration. *Nature reviews neuroscience* **2**, 229–239 (2001).

[31] Schweitzer, F. *et al.* Economic networks: The new challenges. *science* **325**, 422–425 (2009).

[32] Allen, F. & Babus, A. Networks in finance. *The network challenge: strategy, profit, and risk in an interlinked world* **367** (2009).

[33] Cardillo, A., Scellato, S., Latora, V. & Porta, S. Structural properties of planar graphs of urban street patterns. *Physical Review E* **73**, 066107 (2006).

[34] Albert, R. & Barabási, A.-L. Statistical mechanics of complex networks. *Rev. Mod. Phys.* **74**, 47–97 (2002). URL https://link.aps.org/doi/10.1103/RevModPhys.74.47.

[35] Vega-Redondo, F. *Complex social networks*. 44 (Cambridge University Press, 2007).

[36] Borgatti, S. P., Everett, M. G. & Johnson, J. C. *Analyzing social networks* (Sage, 2018).

[37] Faloutsos, M., Faloutsos, P. & Faloutsos, C. On power-law relationships of the internet topology. In *The Structure and Dynamics of Networks*, 195–206 (Princeton University Press, 2011).

[38] Barabási, A.-L. *Network Science* (Cambridge University Press, 2016).

[39] Bang-Jensen, J. & Gutin, G. Z. *Digraphs: theory, algorithms and applications* (Springer Science & Business Media, 2008).

[40] Newman, M. E. Analysis of weighted networks. *Physical review E* **70**, 056131 (2004).

[41] Leskovec, J., Huttenlocher, D. & Kleinberg, J. Signed networks in social media. In *Proceedings of the SIGCHI conference on human factors in computing systems*, 1361–1370 (2010).

[42] Mizuno, T., Takayasu, H. & Takayasu, M. Correlation networks among currencies. *Physica A: Statistical Mechanics and its Applications* **364**, 336–342 (2006).

[43] Chen, Z. J., He, Y., Rosa-Neto, P., Germann, J. & Evans, A. C. Revealing modular architecture of human brain structural networks by using cortical thickness from mri. *Cerebral cortex* **18**, 2374–2381 (2008).

[44] Bachi, G., Coscia, M., Monreale, A. & Giannotti, F. Classifying trust/distrust relationships in online social networks. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, 552–557 (IEEE, 2012).

[45] Carchiolo, V., Longheu, A., Malgeri, M. & Mangioni, G. Trust assessment: a personalized, distributed, and secure approach. *Concurrency and Computation: Practice and Experience* **24**, 605–617 (2012).

[46] Carchiolo, V., Longheu, A., Malgeri, M. & Mangioni, G. Users' attachment in trust networks: reputation vs. effort. *International Journal of Bio-Inspired Computation* **5**, 199–209 (2013).

[47] Newman, M. *Networks: an introduction* (Oxford University Press, 2010).

[48] Watts, D. J. & Strogatz, S. H. Collective dynamics of 'small-world' networks. *nature* **393**, 440–442 (1998).

[49] Newman, M. E. J. Mixing patterns in networks. *Phys. Rev. E* **67**, 026126 (2003). URL https://link.aps.org/doi/10.1103/PhysRevE.67.026126.

[50] Squartini, T., Fagiolo, G. & Garlaschelli, D. Randomizing world trade. I. A binary network analysis. *Physical Review E* **84** (2011). URL http://dx.doi.org/10.1103/PhysRevE.84.046117.

[51] Squartini, T., Fagiolo, G. & Garlaschelli, D. Randomizing world trade. II. A weighted network analysis. *Physical Review E* **84** (2011). URL http://dx.doi.org/10.1103/PhysRevE.84.046118.

[52] Freeman, L. C. A set of measures of centrality based on betweenness. *Sociometry* **40**, 35–41 (1977).

[53] Brandes, U. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks* **30**, 136–145 (2008). URL https://www.sciencedirect.com/science/article/pii/S0378873307000731.

[54] Bonacich, P. Power and centrality: A family of measures. *American journal of sociology* **92**, 1170–1182 (1987).

[55] Newman, M. E. J. *Mathematics of Networks*, 1–8 (Palgrave Macmillan UK, London, 2016). URL https://doi.org/10.1057/978-1-349-95121-5_2565-1.

[56] Perra, N. & Fortunato, S. Spectral centrality measures in complex networks. *Physical Review E* **78**, 036107 (2008).

[57] Page, L., Brin, S., Motwani, R. & Winograd, T. The pagerank citation ranking: Bringing order to the web. Tech. Rep., Stanford InfoLab (1999).

[58] Brin, S. & Page, L. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems* **30**, 107–117 (1998). URL http://www.sciencedirect.com/science/article/pii/S016975529800110X. Proceedings of the Seventh International World Wide Web Conference.

[59] Wasserman, S., Faust, K. *et al.* Social network analysis: Methods and applications (1994).

[60] Radicchi, F., Castellano, C., Cecconi, F., Loreto, V. & Parisi, D. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences* **101**, 2658–2663 (2004). URL https://www.pnas.org/content/101/9/2658. https://www.pnas.org/content/101/9/2658.full.pdf.

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

[62] Blondel, V. D., Guillaume, J.-L., Lambiotte, R. & Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* **2008**, P10008 (2008). URL https://doi.org/10.1088/1742-5468/2008/10/p10008.

[63] Newman, M. E. J. Analysis of weighted networks. *Phys. Rev. E* **70**, 056131 (2004). URL https://link.aps.org/doi/10.1103/PhysRevE.70.056131.

[64] Nicosia, V., Mangioni, G., Carchiolo, V. & Malgeri, M. Extending the definition of modularity to directed graphs with overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment* **2009**, P03024 (2009). URL https://doi.org/10.1088/1742-5468/2009/03/p03024.

[65] Peixoto, T. P. Bayesian stochastic blockmodeling. *Advances in Network Clustering and Blockmodeling* 289–332 (2019). URL http://dx.doi.org/10.1002/9781119483298.ch11.

[66] Peixoto, T. P. Nonparametric bayesian inference of the microcanonical stochastic block model. *Physical Review E* **95** (2017). URL http://dx.doi.org/10.1103/PhysRevE.95.012317.

[67] Holland, P., Laskey, K. B. & Leinhardt, S. Stochastic blockmodels: First steps. *Social Networks* **5**, 109–137 (1983).

[68] Erdös, P. & Rényi, A. On random graphs i. *Publicationes Mathematicae Debrecen* **6**, 290–297 (1959).

[69] Albert, R., Jeong, H. & Barabási, A.-L. Diameter of the world-wide web. *Nature* **401**, 130–131 (1999).

[70] Bianconi, G. & Barabasi, A.-L. Competition and multiscaling in evolving networks. *EPL (Europhysics Letters)* **54**, 436 (2001).

[71] Albert, R., Jeong, H. & Barabási, A.-L. Error and attack tolerance of complex networks. *nature* **406**, 378 (2000).

[72] Crucitti, P., Latora, V. & Marchiori, M. Model for cascading failures in complex networks. *Physical Review E* **69**, 045104 (2004).

[73] Boccaletti, S. *et al.* The structure and dynamics of multilayer networks. *Physics Reports* **544**, 1–122 (2014).

[74] Radicchi, F. Percolation in real interdependent networks. *Nature Physics* **11**, 597 (2015).

[75] Lamberson, P. Diffusion in networks. *The Oxford Handbook of the Economics of Networks* (2016).

[76] De Domenico, M., Solé-Ribalta, A., Gómez, S. & Arenas, A. Navigability of interconnected networks under random failures. *PNAS* **111**, 8351–8356 (2014).

[77] Aleta, A., Meloni, S. & Moreno, Y. A multilayer perspective for the analysis of urban transportation systems. *Scientific reports* **7**, 1–9 (2017).

[78] Dickison, M. E., Magnani, M. & Rossi, L. *Multilayer social networks* (Cambridge University Press, 2016).

[79] Mucha, P. J., Richardson, T., Macon, K., Porter, M. A. & Onnela, J.-P. Community structure in time-dependent, multiscale, and multiplex networks. *Science* **328**, 876–878 (2010).

[80] De Domenico, M. *et al.* Mathematical formulation of multilayer networks. *Physical Review X* **3**, 041022 (2013).

[81] Kivelä, M. *et al.* Multilayer networks. *Journal of complex networks* **2**, 203–271 (2014).

[82] De Domenico, M., Solé-Ribalta, A., Omodei, E., Gómez, S. & Arenas, A. Ranking in interconnected multilayer networks reveals versatile nodes. *Nature Communications* **6**, 6868 (2015).

[83] Bazzi, M. *et al.* Community detection in temporal multilayer networks, with an application to correlation networks. *Multiscale Modeling & Simulation* **14**, 1–41 (2016).

[84] Nicosia, V. & Latora, V. Measuring and modeling correlations in multiplex networks. *Physical Review E* **92**, 032805 (2015).

[85] Gao, J., Buldyrev, S. V., Stanley, H. E. & Havlin, S. Networks formed from interdependent networks. *Nature physics* **8**, 40–48 (2012).

[86] Wang, Z., Wang, L., Szolnoki, A. & Perc, M. Evolutionary games on multilayer networks: a colloquium. *The European physical journal B* **88**, 1–15 (2015).

[87] De Domenico, M., Granell, C., Porter, M. A. & Arenas, A. The physics of spreading processes in multilayer networks. *Nature Physics* **12**, 901–906 (2016).

[88] Gomez, S. *et al.* Diffusion dynamics on multiplex networks. *Physical review letters* **110**, 028701 (2013).

[89] Masuda, N., Porter, M. A. & Lambiotte, R. Random walks and diffusion on networks. *Physics Reports* (2017).

[90] Solá, L. *et al.* Eigenvector centrality of nodes in multiplex networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science* **23**, 033131 (2013).

[91] Cardillo, A. *et al.* Modeling the multi-layer nature of the european air transport network: Resilience and passengers re-scheduling under random failures. *The European Physical Journal Special Topics* **215**, 23–33 (2013).

[92] Battiston, F., Nicosia, V. & Latora, V. Structural measures for multiplex networks. *Physical Review E* **89**, 032804 (2014).

[93] Estrada, E. & Gómez-Gardeñes, J. Communicability reveals a transition to coordinated behavior in multiplex networks. *Physical Review E* **89**, 042819 (2014).

[94] De Domenico, M., Nicosia, V., Arenas, A. & Latora, V. Structural reducibility of multilayer networks. *Nature communications* **6**, 1–9 (2015).

[95] Ghavasieh, A. & De Domenico, M. Enhancing transport properties in interconnected systems without altering their structure. *Physical Review Research* **2**, 013155 (2020).

[96] Bianconi, G. Statistical mechanics of multiplex networks: entropy and overlap. *Physical Review E* **87**, 062806 (2013).

[97] Kolda, T. G. & Bader, B. W. Tensor decompositions and applications. *SIAM Review* **51**, 455–500 (2009).

[98] De Domenico, M., Sasai, S. & Arenas, A. Mapping multiplex hubs in human functional brain networks. *Frontiers in Neuroscience* **10**, 326 (2016). URL https://www.frontiersin.org/article/10.3389/fnins.2016.00326.

[99] Bhattacharya, K., Mukherjee, G., Saramäki, J., Kaski, K. & Manna, S. S. The international trade network: weighted network analysis and modelling. *Journal of Statistical Mechanics: Theory and Experiment* **2008**, P02002 (2008).

[100] Barigozzi, M., Fagiolo, G. & Mangioni, G. Identifying the community structure of the international-trade multi-network. *Physica A: statistical mechanics and its applications* **390**, 2051–2066 (2011).

[101] Mangioni, G., Jurman, G. & De Domenico, M. Multilayer flows in molecular networks identify biological modules in the human proteome. *IEEE Transactions on Network Science and Engineering* **7**, 411–420 (2018).

[102] Choobdar, S. *et al.* Assessment of network module identification across complex diseases. *Nature methods* **16**, 843–852 (2019).

[103] Verstraete, N. *et al.* Covmulnet19, integrating proteins, diseases, drugs, and symptoms: A network medicine approach to covid-19. *Network and systems medicine* **3**, 130–141 (2020).

[104] Cardillo, A. *et al.* Emergence of network features from multiplexity. *Scientific reports* **3**, 1–6 (2013).

[105] Kinsley, A. C., Rossi, G., Silk, M. J. & VanderWaal, K. Multilayer and multiplex networks: An introduction to their use in veterinary epidemiology. *Frontiers in veterinary science* **7**, 596 (2020).

[106] Mitchell, T. M. *Machine Learning* (McGraw-Hill, Inc., New York, NY, USA, 1997), 1 edn.

[107] TURING, A. M. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind* **LIX**, 433–460 (1950). URL https://doi.org/10.1093/mind/LIX.236.433. https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf.

[108] Russell, S. & Norvig, P. *Artificial Intelligence: A Modern Approach* (Prentice Hall Press, USA, 2009), 3rd edn.

[109] Gomez-Uribe, C. A. & Hunt, N. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.* **6** (2016). URL https://doi.org/10.1145/2843948.

[110] Yang, J. *et al.* Improved protein structure prediction using predicted interresidue orientations. *Proceedings of the National Academy of Sciences* **117**, 1496–1503 (2020). URL https://www.pnas.org/content/117/3/1496. https://www.pnas.org/content/117/3/1496.full.pdf.

[111] Jumper, J. *et al.* Highly accurate protein structure prediction with AlphaFold. *Nature* (2021). (Accelerated article preview).

[112] Silver, D. *et al.* Mastering chess and shogi by self-play with a general reinforcement learning algorithm (2017). 1712.01815.

[113] Mnih, V. *et al.* Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).

[114] Edelsten, A. NVIDIA DLSS: Control and Beyond (2019). URL https://www.nvidia.com/en-us/geforce/news/dlss-control-and-beyond/.

[115] Lauri, J. & Dutta, S. Fine-grained search space classification for hard enumeration variants of subset problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2314–2321 (2019).

[116] Murphy, K. P. *Machine Learning: A Probabilistic Perspective* (The MIT Press, 2012).

[117] Goodfellow, I., Bengio, Y. & Courville, A. Deep learning (2016). URL http://www.deeplearningbook.org. Book in preparation for MIT Press.

[118] Pedregosa, F. *et al.* Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011).

[119] Provost, F. & Fawcett, T. *Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking* (O'Reilly Media, 2013). URL https://books.google.ie/books?id=4ZctAAAAQBAJ.

[120] Nielsen, M. A. Neural networks and deep learning (2018). URL http://neuralnetworksanddeeplearning.com/.

[121] Bengio, Y., Courville, A. & Vincent, P. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **35**, 1798–1828 (2013).

[122] Lee, H., Grosse, R., Ranganath, R. & Ng, A. Y. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, 609–616 (Association for Computing Machinery, New York, NY, USA, 2009). URL https://doi.org/10.1145/1553374.1553453.

[123] Hochreiter, S. & Schmidhuber, J. Long Short-Term Memory. *Neural Computation* **9**, 1735–1780 (1997). URL https://doi.org/10.1162/neco.1997.9.8.1735. https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf.

[124] Cho, K. *et al.* Learning phrase representations using rnn encoder-decoder for statistical machine translation (2014). 1406.1078.

[125] Hamilton, W. L. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* **14**, 1–159.

[126] Bronstein, M. M., Bruna, J., Cohen, T. & Veličković, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges (2021). 2104.13478.

[127] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A. & Vandergheynst, P. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine* **34**, 18–42 (2017). URL http://dx.doi.org/10.1109/MSP.2017.2693418.

[128] Bruna, J., Zaremba, W., Szlam, A. & LeCun, Y. Spectral networks and locally connected networks on graphs (2014). 1312.6203.

[129] Dai, H., Dai, B. & Song, L. Discriminative embeddings of latent variable models for structured data (2020). 1603.05629.

[130] Hamilton, W. L., Ying, R. & Leskovec, J. Inductive representation learning on large graphs. *CoRR* **abs/1706.02216** (2017). URL http://arxiv.org/abs/1706.02216. 1706.02216.

[131] Gaudelet, T. *et al.* Utilising graph machine learning within drug discovery and development (2021). 2012.05716.

[132] Zitnik, M., Agrawal, M. & Leskovec, J. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* **34**, 457–466 (2018).

[133] Ying, R. *et al.* Graph convolutional neural networks for web-scale recommender systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018). URL http://dx.doi.org/10.1145/3219819.3219890.

[134] Darvariu, V.-A., Hailes, S. & Musolesi, M. Improving the robustness of graphs through reinforcement learning and graph neural networks (2020). 2001.11279.

[135] Monti, F., Frasca, F., Eynard, D., Mannion, D. & Bronstein, M. M. Fake news detection on social media using geometric deep learning (2019). 1902.06673.

[136] Mirhoseini, A. *et al.* Chip placement with deep reinforcement learning (2020). 2004.10746.

[137] Mirhoseini, A. *et al.* A graph placement methodology for fast chip design. *Nature* **594**, 207–212 (2021).

[138] Chen, Z., Li, X. & Bruna, J. Supervised community detection with line graph neural networks (2020). 1705.08415.

[139] Farrell, S. *et al.* Novel deep learning methods for track reconstruction (2018). 1810.06111.

[140] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. & Dahl, G. E. Neural message passing for quantum chemistry. In Precup, D. & Teh, Y. W. (eds.) *Proceedings of the 34th International Conference on Machine Learning*, vol. 70 of *Proceedings of Machine Learning Research*, 1263–1272 (PMLR, 2017). URL https://proceedings.mlr.press/v70/gilmer17a.html.

[141] Simonovsky, M. & Komodakis, N. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, 412–422 (Springer, 2018).

[142] You, J., Ying, R., Ren, X., Hamilton, W. & Leskovec, J. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, 5708–5717 (PMLR, 2018).

[143] Liao, R. *et al.* Efficient graph generation with graph recurrent attention networks (2020). 1910.00760.

[144] Derrow-Pinion, A. *et al.* Eta prediction with graph neural networks in google maps (2021). 2108.11482.

[145] Zhou, J. *et al.* Graph neural networks: A review of methods and applications. *AI Open* **1**, 57–81 (2020).

[146] Wu, Z. *et al.* A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* (2020).

[147] Rossi, E. *et al.* Temporal graph networks for deep learning on dynamic graphs (2020). 2006.10637.

[148] Sanchez-Gonzalez, A. *et al.* Learning to simulate complex physics with graph networks (2020). 2002.09405.

[149] Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A. & Battaglia, P. W. Learning mesh-based simulation with graph networks (2021). 2010.03409.

[150] Bordes, A., Weston, J., Collobert, R. & Bengio, Y. Learning structured embeddings of knowledge bases. In *Twenty-Fifth AAAI Conference on Artificial Intelligence* (2011).

[151] Hoff, P. D., Raftery, A. E. & Handcock, M. S. Latent space approaches to social network analysis. *Journal of the american Statistical association* **97**, 1090–1098 (2002).

[152] Mikolov, T., Chen, K., Corrado, G. & Dean, J. Efficient estimation of word representations in vector space (2013). 1301.3781.

[153] Tang, D. *et al.* Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1555–1565 (2014).

[154] Belkin, M. & Niyogi, P. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS* (2001).

[155] Cao, S., Lu, W. & Xu, Q. Grarep: Learning graph representations with global structural information 891–900 (2015).

[156] Ou, M., Cui, P., Pei, J., Zhang, Z. & Zhu, W. Asymmetric transitivity preserving graph embedding. In *KDD* (2016).

[157] Grover, A. & Leskovec, J. node2vec: Scalable feature learning for networks. *CoRR* **abs/1607.00653** (2016). URL http://arxiv.org/abs/1607.00653. 1607.00653.

[158] Zitnik, M. & Leskovec, J. Predicting multicellular function through multi-layer tissue networks. *CoRR* **abs/1707.04638** (2017). URL http://arxiv.org/abs/1707.04638. 1707.04638.

[159] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. & Monfardini, G. The graph neural network model. *IEEE transactions on neural networks* **20**, 61–80 (2008).

[160] Li, Y., Tarlow, D., Brockschmidt, M. & Zemel, R. Gated graph sequence neural networks (2017). 1511.05493.

[161] Fey, M. & Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019).

[162] Ying, R. *et al.* Hierarchical graph representation learning with differentiable pooling (2019). 1806.08804.

[163] Kipf, T. N. & Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[164] Veličković, P. *et al.* Graph attention networks (2018). URL https://openreview.net/forum?id=rJXMpikCZ.

[165] Vaswani, A. *et al.* Attention is all you need (2017). 1706.03762.

[166] Li, Q., Han, Z. & Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning (2018). 1801.07606.

[167] Wu, F. *et al.* Simplifying graph convolutional networks (2019). 1902.07153.

[168] Chen, M., Wei, Z., Huang, Z., Ding, B. & Li, Y. Simple and deep graph convolutional networks (2020). 2007.02133.

[169] Derr, T., Ma, Y. & Tang, J. Signed graph convolutional network (2018). 1808.06354.

[170] Xu, D., Ruan, C., Korpeoglu, E., Kumar, S. & Achan, K. Inductive representation learning on temporal graphs (2020). URL https://openreview.net/forum?id=rJeW1yHYwH. 2002.07962.

[171] Ying, R., Bourgeois, D., You, J., Zitnik, M. & Leskovec, J. Gnnexplainer: Generating explanations for graph neural networks (2019). 1903.03894.

[172] Paszke, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. *et al.* (eds.) *Advances in Neural Information Processing Systems 32*, 8024–8035 (Curran Associates, Inc., 2019). URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[173] Battaglia, P. W. *et al.* Relational inductive biases, deep learning, and graph networks (2018). 1806.01261.

[174] Godwin*, J. *et al.* Jraph: A library for graph neural networks in jax. (2020). URL http://github.com/deepmind/jraph.

[175] CSIRO's Data61. Stellargraph machine learning library. https://github.com/stellargraph/stellargraph (2018).

[176] Wang, M. *et al.* Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315* (2019).

[177] Boccaletti, S., Latora, V., Moreno, Y., Chavez, M. & Hwang, D.-U. Complex networks: Structure and dynamics. *Physics reports* **424**, 175–308 (2006).

[178] Barabási, A.-L. & Albert, R. Emergence of scaling in random networks. *science* **286**, 509–512 (1999).

[179] Newman, M. E. Communities, modules and large-scale structure in networks. *Nature physics* **8**, 25–31 (2012).

[180] Fortunato, S. Community detection in graphs. *Physics reports* **486**, 75–174 (2010).

[181] Benson, A. R., Gleich, D. F. & Leskovec, J. Higher-order organization of complex networks. *Science* **353**, 163–166 (2016).

[182] Lambiotte, R., Rosvall, M. & Scholtes, I. From networks to optimal higher-order models of complex systems. *Nature Physics* **15**, 313–320 (2019).

[183] Clauset, A., Moore, C. & Newman, M. E. Hierarchical structure and the prediction of missing links in networks. *Nature* **453**, 98 (2008).

[184] Watts, D. J. & Strogatz, S. H. Collective dynamics of 'small-world' networks. *Nature* **393**, 440–442 (1998). URL http://dx.doi.org/10.1038/30918.

[185] Guimera, R. & Amaral, L. A. N. Functional cartography of complex metabolic networks. *nature* **433**, 895 (2005).

[186] Bassett, D. S. & Sporns, O. Network neuroscience. *Nature neuroscience* **20**, 353 (2017).

[187] Suweis, S., Simini, F., Banavar, J. R. & Maritan, A. Emergence of structural and dynamical properties of ecological mutualistic networks. *Nature* **500**, 449 (2013).

[188] Barthelemy, M. The statistical physics of cities. *Nature Reviews Physics* **1**, 406–415 (2019).

[189] Alves, L. G. A. *et al.* The nested structural organization of the worldwide trade multilayer network. *Scientific Reports* **9**, 2866 (2019). URL https://doi.org/10.1038/s41598-019-39340-w.

[190] Lazer, D. *et al.* Computational social science. *Science* **323**, 721–723 (2009).

[191] Johnson, N. F. *et al.* New online ecology of adversarial aggregates: Isis and beyond. *Science* **352**, 1459–1463 (2016).

[192] Centola, D., Becker, J., Brackbill, D. & Baronchelli, A. Experimental evidence for tipping points in social convention. *Science* **360**, 1116–1119 (2018).

[193] Arenas, A., Díaz-Guilera, A., Kurths, J., Moreno, Y. & Zhou, C. Synchronization in complex networks. *Physics reports* **469**, 93–153 (2008).

[194] Pastor-Satorras, R., Castellano, C., Van Mieghem, P. & Vespignani, A. Epidemic processes in complex networks. *Reviews of modern physics* **87**, 925 (2015).

[195] Matamalas, J. T., Arenas, A. & Gómez, S. Effective approach to epidemic containment using link equations in complex networks. *Science advances* **4**, eaau4212 (2018).

[196] Yang, Y., Nishikawa, T. & Motter, A. E. Small vulnerable sets determine large network cascades in power grids. *Science* **358**, eaan3184 (2017).

[197] Vosoughi, S., Roy, D. & Aral, S. The spread of true and false news online. *Science* **359**, 1146–1151 (2018).

[198] Stella, M., Ferrara, E. & De Domenico, M. Bots increase exposure to negative and inflammatory content in online social systems. *Proceedings of the National Academy of Sciences* **115**, 12435–12440 (2018).

[199] Johnson, N. *et al.* Hidden resilience and adaptive dynamics of the global online hate ecology. *Nature* **573**, 261–265 (2019).

[200] Baronchelli, A. The emergence of consensus: a primer. *Royal Society open science* **5**, 172189 (2018).

[201] Kitsak, M. *et al.* Identification of influential spreaders in complex networks. *Nature physics* **6**, 888 (2010).

[202] Morone, F. & Makse, H. A. Influence maximization in complex networks through optimal percolation. *Nature* **524**, 65 (2015).

[203] Morone, F., Min, B., Bo, L., Mari, R. & Makse, H. A. Collective influence algorithm to find influencers via optimal percolation in massively large social media. *Scientific reports* **6**, 30062 (2016).

[204] Braunstein, A., Dall'Asta, L., Semerjian, G. & Zdeborová, L. Network dismantling. *Proceedings of the National Academy of Sciences* **113**, 12368–12373 (2016). URL http://dx.doi.org/10.1073/pnas.1605083113.

[205] Ren, X.-L., Gleinig, N., Helbing, D. & Antulov-Fantulin, N. Generalized network dismantling. *Proceedings of the National Academy of Sciences* **116**, 6554–6559 (2019).

[206] Buldyrev, S. V., Parshani, R., Paul, G., Stanley, H. E. & Havlin, S. Catastrophic cascade of failures in interdependent networks. *Nature* **464**, 1025 (2010).

[207] Bashan, A., Berezin, Y., Buldyrev, S. V. & Havlin, S. The extreme vulnerability of interdependent spatially embedded networks. *Nature Physics* **9**, 667 (2013).

[208] Osat, S., Faqeeh, A. & Radicchi, F. Optimal percolation on multiplex networks. *Nature communications* **8**, 1540 (2017).

[209] Tremblay, J. *et al.* Deep object pose estimation for semantic robotic grasping of household objects. In *Conference on Robot Learning (CoRL)* (2018). URL https://arxiv.org/abs/1809.10790.

[210] Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B. & Song, L. Learning combinatorial optimization algorithms over graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 6351–6361 (Curran Associates Inc., Red Hook, NY, USA, 2017).

[211] Vaswani, A. *et al.* Attention is all you need. In *Advances in neural information processing systems*, 5998–6008 (2017).

[212] Paszke, A. *et al.* Automatic differentiation in pytorch (2017).

[213] Peixoto, T. P. The graph-tool python library. *figshare* (2014). URL http://figshare.com/articles/graph_tool/1164194.

[214] Csardi, G. & Nepusz, T. The igraph software package for complex network research. *InterJournal* **Complex Systems**, 1695 (2006). URL http://igraph.sf.net.

[215] Hagberg, A. A., Schult, D. A. & Swart, P. J. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, 11–15 (Pasadena, CA USA, 2008).

[216] Clusella, P., Grassberger, P., Pérez-Reche, F. J. & Politi, A. Immunization and targeted destruction of networks using explosive percolation. *Phys. Rev. Lett.* **117**, 208301 (2016). URL https://link.aps.org/doi/10.1103/PhysRevLett.117.208301.

[217] Zdeborová, L., Zhang, P. & Zhou, H.-J. Fast and simple decycling and dismantling of networks. *Scientific Reports* **6** (2016). URL http://dx.doi.org/10.1038/srep37954.

[218] Ribeiro, H. V., Alves, L. G. A., Martins, A. F., Lenzi, E. K. & Perc, M. The dynamical structure of political corruption networks. *Journal of Complex Networks* **6**, 989–1003 (2018). URL https://doi.org/10.1093/comnet/cny002. http://oup.prod.sis.lan/comnet/article-pdf/6/6/989/28007544/cny002.pdf.

[219] Holme, P., Kim, B. J., Yoon, C. N. & Han, S. K. Attack vulnerability of complex networks. *Phys. Rev. E* **65**, 056109 (2002). URL https://link.aps.org/doi/10.1103/PhysRevE.65.056109.

[220] Qi, D. & Majda, A. J. Using machine learning to predict extreme events in complex systems. *PNAS* **117**, 52–59 (2020).

[221] Grassia, M., De Domenico, M. & Mangioni, G. Machine learning dismantling and early-warning signals of disintegration in complex systems (2021). URL https://doi.org/10.5281/zenodo.5105912.

[222] CAIDA. Ipv4 routed /24 as links dataset. URL http://www.caida.org/data/active/ipv4_routed_topology_aslinks_dataset.xml.

[223] Advogato network dataset – KONECT (2017). URL http://konect.cc/networks/advogato.

[224] Massa, P., Salvetti, M. & Tomasoni, D. Bowling alone and trust decline in social network sites. In *Proc. Int. Conf. Dependable, Autonomic and Secure Computing*, 658–663 (2009).

[225] Caenorhabditis elegans network dataset – KONECT (2017). URL http://konect.cc/networks/arenas-meta.

[226] Duch, J. & Arenas, A. Community detection in complex networks using extremal optimization. *Phys. Rev. E* **72**, 027104 (2005).

[227] Google.com internal network dataset – KONECT (2017). URL http://konect.cc/networks/cfinder-google.

[228] Palla, G., Farkas, I. J., Pollner, P., Derényi, I. & Vicsek, T. Directed network modules. *New J. Phys.* **9**, 186 (2007).

[229] Citeseer network dataset – KONECT (2017). URL http://konect.cc/networks/citeseer.

[230] Bollacker, K., Lawrence, S. & Giles, C. L. CiteSeer: An autonomous Web agent for automatic retrieval and identification of interesting publications. In *Proc. Int. Conf. on Autonomous Agents*, 116–123 (1998).

[231] Dblp co-authorship network dataset – KONECT (2017). URL http://konect.cc/networks/com-dblp.

[232] Yang, J. & Leskovec, J. Defining and evaluating network communities based on ground-truth. In *Proc. ACM SIGKDD Workshop on Mining Data Semantics*, 3 (ACM, 2012).

[233] Dblp network dataset – KONECT (2017). URL http://konect.cc/networks/dblp-cite.

[234] Ley, M. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *Proc. Int. Symposium on String Processing and Information Retrieval*, 1–10 (2002).

[235] Digg friends network dataset – KONECT (2017). URL http://konect.cc/networks/digg-friends.

[236] Hogg, T. & Lerman, K. Social dynamics of Digg. *EPJ Data Science* **1** (2012).

[237] Caenorhabditis elegans (neural) network dataset – KONECT (2018). URL http://konect.cc/networks/dimacs10-celegansneural.

[238] White, J. G., Southgate, E., Thomson, J. N. & Brenner, S. The structure of the nervous system of the nematode Caenorhabditis elegans. *Phil. Trans. R. Soc. Lond* **314**, 1–340 (1986).

[239] Political blogs network dataset – KONECT (2018). URL http://konect.cc/networks/dimacs10-polblogs.

[240] Adamic, L. A. & Glance, N. The political blogosphere and the 2004 US election: Divided they blog. In *Proc. Int. Workshop on Link Discov.*, 36–43 (2005).

[241] Douban network dataset – KONECT (2017). URL http://konect.cc/networks/douban.

[242] Zafarani, R. & Liu, H. Social computing data repository at ASU (2009). URL http://socialcomputing.asu.edu.

[243] Rossi, R. A. & Ahmed, N. K. The network data repository with interactive graph analytics and visualization. In *AAAI* (2015). URL http://networkrepository.com.

[244] Twitter lists network dataset – KONECT (2017). URL http://konect.cc/networks/ego-twitter.

[245] McAuley, J. & Leskovec, J. Learning to discover social circles in ego networks. In *Advances in Neural Information Processing Systems*, 548–556 (2012).

[246] Eu institution network dataset – KONECT (2017). URL http://konect.cc/networks/email-EuAll.

[247] Leskovec, J., Kleinberg, J. & Faloutsos, C. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowledge Discovery from Data* **1**, 1–40 (2007).

[248] Matke, C., Medjroubi, W. & Kleinhans, D. SciGRID - An Open Source Reference Model for the European Transmission Network (v0.2) (2016). URL http://www.scigrid.de.

[249] Florida ecosystem dry network dataset – KONECT (2017). URL http://konect.cc/networks/foodweb-baydry.

[250] Ulanowicz, R. E., Heymans, J. J. & Egnotovich, M. S. Network analysis of trophic dynamics in South Florida ecosystems, FY 99: The graminoid ecosystem. *Annual Report to the United States Geological Service Biological Resources Division Ref. No.[UMCES] CBL 00-0176, Chesapeake Biological Laboratory, University of Maryland* (2000).

[251] Florida ecosystem wet network dataset – KONECT (2017). URL http://konect.cc/networks/foodweb-baywet.

[252] Wiegmans, B. Gridkit: European and north-american extracts (2016).

[253] Hyves network dataset – KONECT (2017). URL http://konect.cc/networks/hyves.

[254] Colizza, V., Pastor-Satorras, R. & Vespignani, A. Reaction–diffusion processes and metapopulation models in heterogeneous networks. *Nature Physics* **3**, 276–282 (2007).

[255] Batagelj, V. & Mrvar, A. Pajek datasets. URL http://vlado.fmf.uni-lj.si/pub/networks/data/.

[256] Internet topology network dataset – KONECT (2017). URL http://konect.cc/networks/topology.

[257] Zhang, B., Liu, R., Massey, D. & Zhang, L. Collecting the Internet AS-level topology. *SIGCOMM Computer Communication Review* **35**, 53–61 (2005).

[258] Ciaodvd trust network dataset – KONECT (2018). URL http://konect.cc/networks/librec-ciaodvd-trust.

[259] Guo, G., Zhang, J., Thalmann, D. & Yorke-Smith, N. ETAF: An extended trust antecedents framework for trust prediction. In *Proc. Int. Conf. Adv. in Soc. Netw. Anal. and Min.*, 540–547 (2014).

[260] Filmtrust trust network dataset – KONECT (2018). URL http://konect.cc/networks/librec-filmtrust-trust.

[261] Guo, G., Zhang, J. & Yorke-Smith, N. A novel Bayesian similarity measure for recommender systems. In *Proc. Int. Joint Conf. on Artif. Intell.*, 2619–2625 (2013).

[262] Linux network dataset – KONECT (2017). URL http://konect.cc/networks/linux.

[263] Brightkite network dataset – KONECT (2017). URL http://konect.cc/networks/loc-brightkite_edges.

[264] Cho, E., Myers, S. A. & Leskovec, J. Friendship and mobility: User movement in location-based social networks. In *Proc. Int. Conf. on Knowledge Discovery and Data Mining*, 1082–1090 (2011).

[265] Gowalla network dataset – KONECT (2017). URL http://konect.cc/networks/loc-gowalla_edges.

[266] De Domenico, M., Solé-Ribalta, A., Gómez, S. & Arenas, A. Navigability of interconnected networks under random failures. *Proceedings of the National Academy of Sciences* **111**, 8351–8356 (2014). URL https://www.pnas.org/content/111/23/8351. https://www.pnas.org/content/111/23/8351.full.pdf.

[267] Human protein (stelzl) network dataset – KONECT (2017). URL http://konect.cc/networks/maayan-Stelzl.

[268] Stelzl, U. *et al.* A human protein–protein interaction network: A resource for annotating the proteome. *Cell* **122**, 957–968 (2005).

[269] Human protein (figeys) network dataset – KONECT (2017). URL http://konect.cc/networks/maayan-figeys.

[270] Ewing, R. M. *et al.* Large-scale mapping of human protein–protein interactions by mass spectrometry. *Molecular Systems Biology* **3** (2007).

[271] Little rock lake network dataset – KONECT (2017). URL http://konect.cc/networks/maayan-foodweb.

[272] Martinez, N. D., Magnuson, J. J., Kratz, T. & Sierszen, M. Artifacts or attributes? effects of resolution on the Little Rock Lake food web. *Ecological Monographs* **61**, 367–392 (1991).

[273] Human protein (vidal) network dataset – KONECT (2017). URL http://konect.cc/networks/maayan-vidal.

[274] Rual, J.-F. *et al.* Towards a proteome-scale map of the human protein–protein interaction network. *Nature* 1173–1178 (2005).

[275] Crime network dataset – KONECT (2017). URL http://konect.cc/networks/moreno_crime.

[276] Protein network dataset – KONECT (2017). URL http://konect.cc/networks/moreno_propro.

[277] Coulomb, S., Bauer, M., Bernard, D. & Marsolier-Kergoat, M.-C. Gene essentiality and the topology of protein interaction networks. *Proceedings of the Royal Society B: Biological Sciences* **272**, 1721–1725 (2005).

[278] Han, J.-D. J., Dupuy, D., Bertin, N., Cusick, M. E. & Vidal, M. Effect of sampling on topology predictions of protein-protein interaction networks. *Nature Biotechnology* **23**, 839–844 (2005).

[279] Stumpf, M. P., Wiuf, C. & May, R. M. Subnets of scale-free networks are not scale-free: Sampling properties of networks. *Proceedings of the National Academy of Sciences of the United States of America* **102**, 4221–4224 (2005).

[280] Train bombing network dataset – KONECT (2017). URL http://konect.cc/networks/moreno_train.

[281] Hayes, B. Connecting the dots. can the tools of graph theory and social-network studies unravel the next big plot? *American Scientist* **94**, 400–404 (2006).

[282] Digg network dataset – KONECT (2017). URL http://konect.cc/networks/munmun_digg_reply.

[283] Choudhury, M. D., Sundaram, H., John, A. & Seligmann, D. D. Social synchrony: Predicting mimicry of user actions in online social media. In *Proc. Int. Conf. on Comput. Science and Engineering*, 151–158 (2009).

[284] Twitter (icwsm) network dataset – KONECT (2017). URL http://konect.cc/networks/munmun_twitter_social.

[285] Choudhury, M. D. *et al.* How does the data sampling strategy impact the discovery of information diffusion in social media? In *ICWSM*, 34–41 (2010).

[286] Openflights network dataset – KONECT (2017). URL http://konect.cc/networks/opsahl-openflights.

[287] Opsahl, T., Agneessens, F. & Skvoretz, J. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks* **3**, 245–251 (2010).

[288] Us power grid network dataset – KONECT (2017). URL http://konect.cc/networks/opsahl-powergrid.

[289] Uc irvine messages network dataset – KONECT (2017). URL http://konect.cc/networks/opsahl-ucsocial.

[290] Opsahl, T. & Panzarasa, P. Clustering in weighted networks. *Social Networks* **31**, 155–163 (2009).

[291] Leskovec, J., Kleinberg, J. & Faloutsos, C. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, 177–187 (Association for Computing Machinery, New York, NY, USA, 2005). URL https://doi.org/10.1145/1081870.1081893.

[292] Ripeanu, M., Foster, I. & Iamnitchi, A. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Comput. J.* **6** (2002).

[293] Leskovec, J. & Krevl, A. SNAP Datasets: Stanford large network dataset collection (2014). URL http://snap.stanford.edu/data.

[294] Gnutella network dataset – KONECT (2017). URL http://konect.cc/networks/p2p-Gnutella31.

[295] Erdős network dataset – KONECT (2018). URL http://konect.cc/networks/pajek-erdos.

[296] Catster/dogster familylinks/friendships network dataset – KONECT (2017). URL http://konect.cc/networks/petster-carnivore.

[297] Hamsterster full network dataset – KONECT (2017). URL http://konect.cc/networks/petster-hamster.

[298] Li, F., Cheng, D., Hadjieleftheriou, M., Kollios, G. & Teng, S.-H. On trip planning queries in spatial databases. In Bauzer Medeiros, C., Egenhofer, M. J. & Bertino, E. (eds.) *Advances in Spatial and Temporal Databases*, 273–290 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2005).

[299] Penn State University Libraries. Digital chart of the world server (2006). URL http://www.maproom.psu.edu/dcw/.

[300] Brinkhoff, T. A framework for generating network-based moving objects. *GeoInformatica* **6** (2000).

[301] Route views network dataset – KONECT (2017). URL http://konect.cc/networks/as20000102.

[302] Slashdot threads network dataset – KONECT (2017). URL http://konect.cc/networks/slashdot-threads.

[303] Gómez, V., Kaltenbrunner, A. & López, V. Statistical analysis of the social network and discussion threads in Slashdot. In *Proc. Int. World Wide Web Conf.*, 645–654 (2008).

[304] Slashdot zoo network dataset – KONECT (2017). URL http://konect.cc/networks/slashdot-zoo.

[305] Kunegis, J., Lommatzsch, A. & Bauckhage, C. The Slashdot Zoo: Mining a social network with negative edges. In *Proc. Int. World Wide Web Conf.*, 741–750 (2009). URL http://uni-koblenz.de/~kunegis/paper/kunegis-slashdot-zoo.pdf.

[306] Jdk dependency network dataset – KONECT (2016). URL http://konect.cc/networks/subelj_jdk.

[307] Jung and javax dependency network dataset – KONECT (2017). URL http://konect.cc/networks/subelj_jung-j.

[308] Šubelj, L. & Bajec, M. Software systems through complex networks science: Review, analysis and applications. In *Proc. Int. Workshop on Software Mining*, 9–16 (2012).

[309] De Nooy, W., Mrvar, A. & Batagelj, V. *Exploratory social network analysis with Pajek*, vol. 27 (Cambridge University Press, 2011).

[310] Notre dame network dataset – KONECT (2017). URL http://konect.cc/networks/web-NotreDame.

[311] Stanford network dataset – KONECT (2017). URL http://konect.cc/networks/web-Stanford.

[312] Leskovec, J., Lang, K., Dasgupta, A. & Mahoney, M. W. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* **6**, 29–123 (2009).

[313] Boldi, P., Codenotti, B., Santini, M. & Vigna, S. UbiCrawler: A scalable fully distributed web crawler. *Software: Practice & Experience* **34**, 711–726 (2004).

[314] Boldi, P., Rosa, M., Santini, M. & Vigna, S. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW*, 587–596 (2011).

[315] Wikipedia links (li) network dataset – KONECT (2018). URL http://konect.cc/networks/wikipedia_link_li.

[316] Wikipedia links (kn) network dataset – KONECT (2018). URL http://konect.cc/networks/wikipedia_link_kn.

[317] Wordnet network dataset – KONECT (2017). URL http://konect.cc/networks/wordnet-words.

[318] Fellbaum, C. (ed.) *WordNet: an Electronic Lexical Database* (MIT Press, 1998).

[319] Muscoloni, A., Thomas, J. M., Ciucci, S., Bianconi, G. & Cannistraci, C. V. Machine learning meets complex networks via coalescent embedding in the hyperbolic space. *Nature communications* **8**, 1–19 (2017).

[320] Boguna, M., Krioukov, D. & Claffy, K. C. Navigability of complex networks. *Nature Physics* **5**, 74–80 (2009).

[321] da F. Costa, L., Rodrigues, F. A., Travieso, G. & Boas, P. R. V. Characterization of complex networks: A survey of measurements. *Advances in Physics* **56**, 167–242 (2007). URL https://doi.org/10.1080/00018730601170527. https://doi.org/10.1080/00018730601170527.

[322] Latora, V., Nicosia, V. & Russo, G. *Complex Networks: Principles, Methods and Applications* (Cambridge University Press, USA, 2017).

[323] Borgatti, S. P. & Everett, M. G. A graph-theoretic perspective on centrality. *Social Networks* **28**, 466–484 (2006). URL https://www.sciencedirect.com/science/article/pii/S0378873305000833.

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

[325] Das, K., Samanta, S. & Pal, M. Study on centrality measures in social networks: a survey. *Social Network Analysis and Mining* **8** (2018).

[326] Carchiolo, V., Longheu, A., Malgeri, M. & Mangioni, G. The cost of trust in the dynamics of best attachment. *Computing & Informatics* **34** (2015).

[327] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. *Climbing Ranking Position via Long-Distance Backlinks: 11th International Conference, IDCS 2018, Tokyo, Japan, October 11-13, 2018, Proceedings*, 100–108 (Springer International Publishing, 2018).

[328] Carchiolo, V., Longheu, A., Malgeri, M. & Mangioni, G. Gain the best reputation in trust networks. In Brazier, F., Nieuwenhuis, K., Pavlin, G., Warnier, M. & Badica, C. (eds.) *Intelligent Distributed Computing V*, vol. 382 of *Studies in Computational Intelligence*, 213–218 (Springer Berlin Heidelberg, 2012). URL http://dx.doi.org/10.1007/978-3-642-24013-3_21.

[329] Olsen, M. & Viglas, A. On the approximability of the link building problem. *Theor. Comput. Sci.* **518**, 96–116 (2014). URL http://dx.doi.org/10.1016/j.tcs.2013.08.003.

[330] Olsen, M. The computational complexity of link building. In Hu, X. & Wang, J. (eds.) *Computing and Combinatorics*, 119–129 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008).

[331] Olsen, M. Maximizing pagerank with new backlinks. In Calamoneri, T. & Diaz, J. (eds.) *Algorithms and Complexity*, 37–48 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2010).

[332] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A. & Vandergheynst, P. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine* **34**, 18–42 (2017).

[333] Lee, J. B., Rossi, R. A., Kim, S., Ahmed, N. K. & Koh, E. Attention models in graphs: A survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)* **13**, 1–25 (2019).

[334] ©Google. Facts about google and competition. URL https://web.archive.org/web/20111104131332/https://www.google.com/competition/howgooglesearchworks.html.

[335] Richardson, M. & Domingos, P. The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank. In *Advances in Neural Information Processing Systems 14* (MIT Press, 2002). URL http://citeseer.ist.psu.edu/460350.html.

[336] Kamvar, S., Kamvar, A., Haveliwala, T. & Golub, G. Adaptive methods for the computation of pagerank. Tech. Rep., Stanford University (2003).

[337] Bianchini, M., Gori, M. & Scarselli, F. Inside pagerank. *ACM Trans. Internet Technol.* **5**, 92–128 (2005). URL http://doi.acm.org/10.1145/1052934.1052938.

[338] Berkhin, P. A survey on pagerank computing. *Internet Mathematics* **2**, 73–120 (2005). URL http://www.projecteuclid.org/DPubS?verb=Display&version=1.0&service=UI&handle=euclid.im/1128530802&page=record.

[339] Buzzanca, M., Carchiolo, V., Longheu, A., Malgeri, M. & Mangioni, G. Black hole metric: Overcoming the pagerank normalization problem. *CoRR* **abs/1802.05453** (2018). URL http://arxiv.org/abs/1802.05453. 1802.05453.

[340] Csáji, B. C., Jungers, R. M. & Blondel, V. D. Pagerank optimization by edge selection. *CoRR* **abs/0911.2280** (2009).

[341] de Kerchove, C., Ninove, L. & Dooren, P. V. Maximizing pagerank via outlinks. *CoRR* **abs/0711.2867** (2007).

[342] Zhirov, A. O., Zhirov, O. V. & Shepelyansky, D. L. Two-dimensional ranking of wikipedia articles. *CoRR* **abs/1006.4270** (2010). URL http://arxiv.org/abs/1006.4270.

[343] Roa-Valverde, A. J. & Sicilia, M.-A. A survey of approaches for ranking on the web of data. *Inf. Retr.* **17**, 295–325 (2014). URL http://dx.doi.org/10.1007/s10791-014-9240-0.

[344] Gupta, P. *et al.* Wtf: The who to follow service at twitter. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, 505–514 (International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 2013). URL http://dl.acm.org/citation.cfm?id=2488388.2488433.

[345] Carchiolo, V., Longheu, A. & Malgeri, M. Reliable peers and useful resources: Searching for the best personalised learning path in a trust- and recommendation-aware environment. *Information Sciences* **180**, 1893–1907 (2010). Special Issue on Intelligent Distributed Information Systems.

[346] Serrano-Guerrero, J., Romero, F. & Olivas, J. Hiperion: A fuzzy approach for recommending educational activities based on the acquisition of competences. *Information Sciences* – (2013).

[347] Jiang, B., Zhao, S. & Yin, J. Self-organized natural roads for predicting traffic flow: a sensitivity study. *Journal of Statistical Mechanics: Theory and Experiment* **2008**, P07008 (2008). URL http://dx.doi.org/10.1088/1742-5468/2008/07/P07008.

[348] Chen, L., Chen, G. & Wang, F. Recommender systems based on user reviews: The state of the art. *User Modeling and User-Adapted Interaction* **25**, 99–154 (2015). URL http://dx.doi.org/10.1007/s11257-015-9155-5.

[349] Carchiolo, V., Longheu, A., Malgeri, M. & Mangioni, G. Searching for experts in a context-aware recommendation network. *Computers in Human Behavior* **51**, 1086–1091 (2015). URL http://www.sciencedirect.com/science/article/pii/S0747563215002186. Computing for Human Learning, Behaviour and Collaboration in the Social and Mobile Networks Era.

[350] Gleich, D. F. Pagerank beyond the web. *CoRR* **abs/1407.5107** (2014). URL http://arxiv.org/abs/1407.5107. 1407.5107.

[351] Avrachenkov, K. & Litvak, N. The effect of new links on google pagerank. *Stochastic Models* **22**, 319–331 (2006). URL http://doc.utwente.nl/63648/.

[352] Olsen, M., Viglas, A. & Zvedeniouk, I. An approximation algorithm for the link building problem. *CoRR* **abs/1204.1369** (2012). URL http://arxiv.org/abs/1204.1369.

[353] Carchiolo, V., Longheu, A., Malgeri, M. & Mangioni, G. A heuristic to explore trust networks dynamics. In Zavoral, F., Jung, J. J. & Badica, C. (eds.) *Intelligent Distributed Computing VII*, 67–76 (Springer International Publishing, Cham, 2014).

[354] Carchiolo, V., Grassia, M., Longheu, A., Malgeri, M. & Mangioni, G. Long distance in-links for ranking enhancement. In Del Ser, J. *et al.* (eds.) *Intelligent Distributed Computing XII*, 3–10 (Springer International Publishing, Cham, 2018).

[355] Marsh, S. Formalising trust as a computational concept. Tech. Rep., University of Stirling (1994). PhD thesis.

[356] Walter, F. E., Battiston, S. & Schweitzer, F. A model of a trust-based recommendation system on a social network. *JOURNAL OF AUTONOMOUS AGENTS AND MULTI-AGENT SYSTEMS* **16**, 57 (2008). URL doi:10.1007/s10458-007-9021-x.

[357] Buzzanca, M., Carchiolo, V., Longheu, A., Malgeri, M. & Mangioni, G. Dealing with the best attachment problem via heuristics. In Badica, C. e. a. (ed.) *Intelligent Distributed Computing X*, vol. 678, 205–214 (Springer International Publishing, Cham, 2017).

[358] Carchiolo, V., Longheu, A., Malgeri, M. & Mangioni, G. The effect of topology on the attachment process in trust networks. In *Intelligent Distributed Computing VIII*, 377–382 (Springer, 2015).

[359] Dorogovtsev, S. N., Goltsev, A. V. & Mendes, J. F. F. k-core organization of complex networks. *Phys. Rev. Lett.* **96**, 040601 (2006).

[360] Goh, K.-I., Kahng, B. & Kim, D. Universal behavior of load distribution in scale-free networks. *Physical Review Letters* **87** (2001). URL http://dx.doi.org/10.1103/PhysRevLett.87.278701.

[361] arxiv hep-ph network dataset – KONECT (2017). URL http://konect.cc/networks/cit-HepPh.

[362] arxiv hep-th network dataset – KONECT (2017). URL http://konect.cc/networks/cit-HepTh.

[363] Google+ (nips) network dataset – KONECT (2017). URL http://konect.cc/networks/ego-gplus.

[364] Foldoc network dataset – KONECT (2017). URL http://konect.cc/networks/foldoc.

[365] Batagelj, V., Mrvar, A. & ZaveÅ¡nik, M. Network analysis of texts. In *Language Technologies*, 143–148 (2002).

[366] Celli, F., Lascio, F. M. L. D., Magnani, M., Pacelli, B. & Rossi, L. Social Network Data and Practices: the case of Friendfeed. In *International Conference on Social Computing, Behavioral Modeling and Prediction*, Lecture Notes in Computer Science (Springer Berlin Heidelberg, 2010).

[367] Magnani, M. & Rossi, L. The ML-Model for Multi-layer Social Networks. In *ASONAM*, 5–12 (IEEE Computer Society, 2011).

[368] Blogs network dataset – KONECT (2017). URL http://konect.cc/networks/moreno_blogs.

[369] Adolescent health network dataset – KONECT (2017). URL http://konect.cc/networks/moreno_health.

[370] Moody, J. Peer influence groups: Identifying dense clusters in large networks. *Soc. Netw.* **23**, 261–283 (2001).

[371] Openflights (patokallio) network dataset – KONECT (2017). URL http://konect.cc/networks/openflights.

[372] Physicians network dataset – KONECT (2017). URL http://konect.cc/networks/moreno_innovation.

[373] Coleman, J., Katz, E. & Menzel, H. The diffusion of an innovation among physicians. *Sociometry* 253–270 (1957).

[374] Epinions network dataset – KONECT (2017). URL http://konect.cc/networks/soc-Epinions1.

[375] Richardson, M., Agrawal, R. & Domingos, P. Trust management for the semantic web. In *Proc. Int. Semant. Web Conf.*, 351–368 (2003).

[376] Bitcoin otc network dataset – KONECT (2018). URL http://konect.cc/networks/soc-sign-bitcoinotc.

[377] Kumar, S., Spezzano, F., Subrahmanian, V. S. & Faloutsos, C. Edge weight prediction in weighted signed networks. In *Proc. Int. Conf. Data Min.*, 221–230 (2016).

[378] Cora network dataset – KONECT (2017). URL http://konect.cc/networks/subelj_cora.

[379] Šubelj, L. & Bajec, M. Model of complex networks based on citation dynamics. In *Proc. of the WWW Workshop on Large Scale Network Analysis*, 527–530 (2013).

[380] Wikipedia elections network dataset – KONECT (2017). URL http://konect.cc/networks/elec.

[381] Leskovec, J., Huttenlocher, D. & Kleinberg, J. Governance in social media: A case study of the Wikipedia promotion process. In *Proc. Int. Conf. on Weblogs and Soc. Media* (2010).

[382] Paszke, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. *et al.* (eds.) *Advances in Neural Information Processing Systems 32*, 8024–8035 (Curran Associates, Inc., 2019). URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[383] Sole-Ribalta, A. *et al.* Spectral properties of the laplacian of multiplex networks. *Physical Review E* **88**, 032807 (2013).

[384] Li, J., Chen, C., Tong, H. & Liu, H. *Multi-Layered Network Embedding*, 684–692 (2018).

[385] Ghorbani, M., Baghshah, M. S. & Rabiee, H. R. Mgcn. *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (2019). URL http://dx.doi.org/10.1145/3341161.3342942.

[386] Khan, M. R. & Blumenstock, J. E. Multi-gcn: Graph convolutional networks for multi-view networks, with applications to global poverty (2019). 1901.11213.

[387] Larremore, D. B., Clauset, A. & Buckee, C. O. A network approach to analyzing highly recombinant malaria parasite genes. *PLOS Computational Biology* **9**, 1–12 (2013). URL https://doi.org/10.1371/journal.pcbi.1003268.

[388] Gómez, S. *et al.* Diffusion dynamics on multiplex networks. *Phys. Rev. Lett.* **110**, 028701 (2013). URL https://link.aps.org/doi/10.1103/PhysRevLett.110.028701.

[389] Leli, V. M., Osat, S., Tlyachev, T., Dylov, D. & Biamonte, J. Deep learning super-diffusion in multiplex networks. *Journal of Physics: Complexity* (2021). URL http://iopscience.iop.org/article/10.1088/2632-072X/abe6e9.

[390] Massa, P., Salvetti, M. & Tomasoni, D. Bowling alone and trust decline in social network sites. *2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing* 658–663 (2009).

[391] Kumar, S., Spezzano, F., Subrahmanian, V. S. & Faloutsos, C. Edge weight prediction in weighted signed networks. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 221–230 (2016).

[392] Massa, P. & Avesani, P. Controversial users demand local trust metrics: an experimental study on epinions.com community. In *Proc. American Association for Artif. Intell. Conf.*, 121–126 (2005).

[393] Kunegis, J. *et al. Spectral Analysis of Signed Graphs for Clustering, Prediction and Visualization*, 559–570. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611972801.49. https://epubs.siam.org/doi/pdf/10.1137/1.9781611972801.49.

[394] Kipf, T. N. & Welling, M. Semi-supervised classification with graph convolutional networks (2017). 1609.02907.

List of figures

List of tables