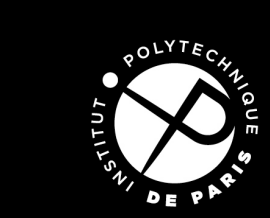
Etude de Dégénérescence de Graph appliqué à l'Apprentissage Automatique Avancé et Résultats Théoriques relatifs

Stratis Limnios



INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2020IPPAX038

Thèse de doctorat

Graph Degeneracy Studies for Advanced Learning Methods on Graphs and Theoretical Results

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'Ecole polytechnique

École doctorale n°626 Institut Polytechnique de Paris (ED IPP)
Spécialité de doctorat : Mathématiques et Informatique

Thèse présentée et soutenue à Paris, le 31/08/2020, par

STRATIS LIMNIOS

Composition du Jury :

Michalis Vazirgiannis Professor, Ecole Polytechnique, Palaiseau, France	Directeur de thèse
Dimitrios Thilikos CNRS, University of Montpellier, France	Co-directeur de thèse
Jean-Philippe Vert	Président du Jury
Professor, Mines ParisTech, Paris, France	Rapporteur
William L. Hamilton Assistant Professor, McGill University, Montreal, Canada	Rapporteur
Eunjung Kim Research Fellow, CNRS, University Paris Dauphine, France	Examinateur
Jie Tang Professor, Tsinghua University, Beijing, China	Examinateur
Michal Valko Research Fellow, Inria Lille, France	Examinateur
Efstratios Gallopoulos Professor, University of Patras, Greece	Examinateur

Abstract

Network science has emerged over the last decade as a field of interest for understanding behaviors and has been the core of many advances in the field of machine learning. Indeed, data mining algorithms were destined for structured/relational data while many datasets exist that require graph representation such as social networks, networks generated by textual data, $3D$ protein structures and chemical compounds, etc. It is henceforth of crucial importance to be able to extract information from graphs in an effective and efficient way.

In this thesis we aim to capitalize on the combinatorial properties of graphs to provide meaningful substructures and decompositions, namely studying graph degeneracy. As degeneracy and admissibility are notions already thoroughly studied in the field of theoretical graph combinatorics and analysis, they started to appear in data mining frameworks, such as the k-core and its variants. These applications have already proven to be state-of-the-art methods in several unsupervised learning scenarios and are considered as milestones in graph mining, for community detection, anomaly and fraud detection, and finding influential spreaders.

In the first part of the thesis, we study edge-degeneracy, which is the edge counterpart of the classic degeneracy. We use a cops and robbers game where the cops are blocking edges of a graph, while the robber occupies its vertices, to extract as the capture cost of the robber the hierarchy of invariants that is the k-edge-degeneracy. We then provide equivalent definitions and prove a min-max theorem that supports this equivalence. Furthermore, as a consequence of this theorem, we can identify the computational complexity of s-edge-degeneracy : it can be computed in polynomial time when $s \in \{1, 2, \infty\}$, while for all other values of s, deciding whether its value is at most k is an NP-complete problem. Finally we prove as well a structural theorem for the ∞-edge-degeneracy, called edge-admissibility, stating that a graph with bounded edge-admissibility can be reconstructed in a specific way from graphs with almost-edge-bounded degree.

The second part of the thesis aims to propose a unified degeneracy and admissibility framework. We propose a way to generate degree as well as connectivity degeneracy and admissibility hierarchies from eight different graph metrics ; among them, four are vertex driven metrics and the other four are edge driven ones. We provide as well adapted orderings comparing degree and connectivity based degeneracies. From these decompositions we choose to apply directly to real-world graphs the most promising one, *i.e.* k-edge-connectivity degeneracy. As we proved some conditions on the complexity of algorithms computing it, we designed such an algorithm to find the k-edge-connectivity cores of a graph. As the edge connectivity counterpart of the k-core has better connectivity properties, it is only natural to wonder if it can outperform it in standard tasks where the k-core and its variants are competitive. Unfortunately, even though we proved that our algorithm is polynomial in terms of computational complexity, the results were not convincing enough regarding the gain of performance for tasks such as finding dense subgraphs and better influential spreaders compared to the loss in runtime, as for large networks the algorithm could run in days compared to seconds for the k-core.

Facing these results, two conclusions lead us to our next contribution. Even though there are theoretical advantages for using the edge-degeneracy, when applied to real-world datasets, the difference with the k-core is not very significant. Moreover, the k-core is a very efficient and effective method. Hence, capitalizing on those observations we tackled another field of application of graph learning where degeneracy was never tested : graph similarity. The problem of accurately measuring the similarity between graphs is at the core of many applications in a variety of disciplines. Most of the existing methods for graph similarity, focuses either on local or on the global properties of graphs. However, even if graphs seem very similar from a local or a global perspective, they may exhibit different structures at different scales. We provide a general framework that uses the k-core decomposition of a graph to produce a hierarchy of nested subgraphs where we apply state of the art kernels. We were also

able to provide a theoretical guarantee in the means of a probabilistic inequality, depending on the k-core of the graph and the eigenvalues of the kernel matrix, that if satisfied it is almost certain that the k-core version of the kernel will outperform the original one.

Finally, in the last part of the thesis, since the results of our previous contribution were very encouraging, we thought about taking another step forward into tackling a core framework in machine learning, bringing the k-core to the deep learning field. Indeed, as a neural network is a multipartite graph, we wanted to analyze directly its structure to extract meaningful information from the learning itself of these algorithms. We first provided an adapted version of the k-core for the graph, extracted from the neural network. Hence, considering it as a complete weighted multipartite graph, we designed a hypergraph version of the k-core. Precisely, the obtained bipartite graph is a representation of a hypergraph : the nodes on one side are the hyperedges, on the other side, the hypernodes and the links enhance which hypernode belongs to which hyperedge. Equipped with the latter representation, we were able to run this hypercore to several neural network architectures. More specifically, applications to convolutional neural networks and multi-layer perceptrons for image recognition, after a very small training, were derived. Finally, we used the information provided by the core numbers of the neurons to re-initialize the weights of the neural network to give a more specific direction for the gradient optimization scheme. This method was able to outperform the original architectures with state-of-the-art initialization methods for most of the activation fonctions.

Résumé

L'analyse des réseaux est apparue au cours de la dernière décennie comme un domaine d'intérêt pour la compréhension de comportements et a été au cœur de nombreuses avancées en apprentissage automatique. En effet, les algorithmes de fouille de données étaient destinés à des données structurées/ relationnelles, alors que de nombreux ensembles de données nécessitent une représentation par graphe, tels que par exemples les réseaux sociaux, les réseaux générés par des données textuelles, les structures protéiques en trois dimensions, les composés chimiques, etc. Il est désormais d'une importance cruciale de pouvoir extraire l'information des graphes de manière efficace et efficiente.

Dans cette thèse, nous visons à capitaliser sur les propriétés combinatoires des graphes pour fournir des sous-structures et des décompositions significatives, notamment en étudiant la dégénérescence des graphes. En effet, comme la dégénérescence et l'admissibilité sont des notions déjà bien étudiées dans le domaine de la combinatoire théorique et l'analyse des graphes, elles ont commencé à apparaître dans les cadres d'exploration de données, comme le k-core et ses variations. Ces applications se sont déjà révélées être des méthodes à l'état de l'art dans plusieurs scénarios d'apprentissage non supervisés et sont considérées comme des étapes importantes dans l'exploration des graphes, pour la détection des communautés, la détection des anomalies et des fraudes, et la recherche de sources d'influences.

Dans la première partie de la thèse, nous étudions la dégénérescence d'arêtes, qui est analogue à la dégénérescence classique. Nous utilisons un jeu de policiers et de voleurs, où les policiers bloquent les arêtes d'un graphe, tandis que le voleur occupe ses sommets, et ainsi extraire comme coût de capture du voleur la hiérarchie d'invariants, qui est la k-arête-dégénérescence. Nous fournissons ensuite des définitions équivalentes et prouvons un théorème min-max qui soutient cette équivalence. En outre, grâce à ce théorème, nous pouvons identifier la complexité de calcul de la dégénérescence de s-arête : elle peut être calculée en temps polynomial lorsque $s \in \{1, 2, \infty\}$, tandis que pour toutes

les autres valeurs de s, décider si sa valeur est au plus k est un problème NP-complet. Enfin, nous démontrons également un théorème structurel pour la ∞-arête-dégénérescence, appelée arête-admissibilité, selon lequel un graphe avec arête-admissibilité limité peut être reconstruit d'une manière spécifique, à partir de graphe avec degré d'arêtes quasiment borné.

La seconde partie de la thèse se voue à proposer un cadre unificateur pour, à la fois, la dégénérescence et l'admissibilité. Nous proposons une méthode pour générer les hiérarchies de dégénérescence de connectivité et de degré tirés de huit métriques de graphes ; parmi elle, quatre découlent de métriques de nœuds et les quatre autres découlent de métriques d'arêtes. Nous proposons de plus un ordre de comparaison entre les dégénérescence de degré et de connectivité. Parmi ces décompositions, nous choisissons d'appliquer la hiérarchie la plus prometteuse à des problématiques réelles, *i.e.* la k-arête-connectivité dégénérescence. Ayant prouvé certaines conditions sur la complexité des algorithmes l'évaluant, nous désignons donc un tel algorithme renvoyant la décomposition en cores de k-arête-connectivité du graphe. La deuxième partie de la thèse vise à appliquer directement aux graphes du monde réel la k-arête-dégénérescence. Comme nous avons démontré certaines conditions sur la complexité des algorithmes qui la calculent, nous avons conçu un algorithme similaire pour trouver les k-arêtes-cores d'un graphe. Étant donné que l'analogue en termes d'arêtes du k-core a de meilleures propriétés de connectivité, il est donc naturel de vérifier si cette première est capable de dépasser la performance des k-core et ses variantes, dans les taches classiques où brillent ces derniers. Malheureusement, même si nous avons prouvé que notre algorithme est polynomial en termes de complexité de calcul, les résultats n'ont pas été assez convaincants. En particulier, en ce qui concerne le gain de performance pour des tâches telles que la recherche de sous-graphes denses ou des meilleurs influenceurs, mais aussi en terme de temps d'exécution, où pour les grands réseaux, l'algorithme pourrait fonctionner en quelques jours au lieu de quelques secondes pour le k-core.

Face à ces résultats, deux conclusions nous amènent à notre prochaine contribution. Malgré les avantages théoriques à l'utilisation de la dégéné-

rescence d'arêtes, lorsqu'elle est appliquée à des ensembles de données réelles, la différence avec le k-core n'est pas très significative. De plus, la méthode k-core est très efficace et efficiente. C'est pourquoi, en capitalisant sur ces observations, nous avons abordé un autre domaine d'application de l'apprentissage des graphes où la dégénérescence n'a jamais été testée : la similarité des graphes. Le problème de la mesure précise de la similarité entre les graphes est au cœur de nombreuses applications dans diverses disciplines. La plupart des méthodes existantes pour la similarité des graphes se concentrent soit sur les propriétés locales, soit sur les propriétés globales des graphes. Cependant, même si les graphes semblent très similaires d'un point de vue local ou global, ils peuvent présenter des structures différentes à des échelles différentes. Nous fournissons un cadre général qui utilise la décomposition en k-core d'un graphe pour produire une hiérarchie de sous-graphes imbriqués, où nous appliquons des noyaux de l'état de l'art. Nous avons également pu fournir une garantie théorique dans les moyens d'une inégalité probabiliste, en fonction du k-core du graphe et des valeurs propres de la matrice du noyau, que si elle est satisfaite, il est presque certain que la version k-core du noyau surpassera la version originale.

Enfin, dans la dernière partie de la thèse, et au regard des résultats très encourageants de cette dernière contribution, nous avons pensé à faire un pas de plus pour aborder un cadre de base dans l'apprentissage automatique, en portant le k-core au domaine de l'apprentissage profond. En effet, comme un réseau de neurones est un graphe multipartie, nous voulons analyser directement sa structure pour en extraire des informations significatives de l'apprentissage même de ces algorithmes. Nous avons d'abord fourni une version adaptée du k-core pour le graphe extrait du réseau de neurones. Comme il s'agit d'un graphe multipartie complet et pondéré, nous avons conçu une version du k-core adapté aux hypergraphes. En effet, un graphe bipartie est une représentation d'un hypergraphe : les nœuds d'un côté étant les hyperarêtes, de l'autre côté, les hypernœuds et les liens montrant quel hypernœud appartient à quelle hyperarête. Nous avons ensuite pu faire fonctionner cet hypercore sur plusieurs architectures de réseaux de neurones, et plus spécifiquement sur

des réseaux neuronaux convolutifs et des perceptrons multicouches pour la reconnaissance d'images après un très petit temps d'entraînement. Pour finir, nous avons ensuite utilisé les informations fournies par l'hypercore d'appartenance des neurones pour réinitialiser les poids du réseau de neurones afin de donner une direction plus spécifique au schéma d'optimisation du gradient. Cette méthode a permis de surpasser les architectures originales grâce à des méthodes d'initialisation pour la plupart des fonctions d'activation.

List of Publications

The following publications and submissions are included in parts or in an extended version of this thesis :

— Limnios, S., Paul, C., Perret, J., and Thilikos, D. M. (2020b). Edge degeneracy: Algorithmic and structural results. *Theoretical Computer Science*

— Limnios, S., Thilikos, D., and Vazirgiannis, M. (2018). Degeneracy Hierarchy Generator and Efficient Connectivity Degeneracy Algorithm. *not published*

— Nikolentzos, G., Meladianos, P., Limnios, S., and Vazirgiannis, M. (2018). A Degeneracy Framework for Graph Similarity. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2595–2601. International Joint Conferences on Artificial Intelligence Organization

— Skianis, K., Nikolentzos, G., Limnios, S., and Vazirgiannis, M. (2020). Rep the set: Neural networks for learning set representations. In *International Conference on Artificial Intelligence and Statistics*, pages 1410–1420
Paper accepted at the International Conference on Artificial Intelligence and Statistics (AISTATS) 2020.

— Limnios, S., Dasoulas, G., Thilikos, D. M., and Vazirgiannis, M. (2020a). Hcore-init: Neural network initialization based on graph degeneracy. *arXiv*

preprint arXiv:2004.07636
Accepted at the Internationnal Conference on Pattern Recognition (ICPR) 2020.

In addition to these publications, the following are also part of my work : the production of a library for graph kernels and a paper concerning link prediction in directed graphs using auto-encoders. As they do not fall under the scope of this thesis they are not covered in the dissertation :

— Siglidis, G., Nikolentzos, G., Limnios, S., Giatsidis, C., Skianis, K., and Vazirgiannis, M. (2020). Grakel: A graph kernel library in python. *Journal of Machine Learning Research*, 21(54):1–5

— Salha, G., Limnios, S., Hennequin, R., Tran, V. A., and Vazirgiannis, M. (2019b). Gravity-inspired graph autoencoders for directed link prediction. In *CIKM '19*

Table des matières

Table des figures

Liste des tableaux

CHAPITRE 1

Introduction

Networks are already ubiquitous as data form from many disciplines. *Relationship Networks* are the most obvious domain where graph data shines, as they provide meaningful information to model and understand social interactions. Indeed the need to analyse academic collaborations [Tang et al., 2008], e-mail exchange networks and in a more general fashion, social network analysis [Leskovec et al., 2009], is at the core of graph mining, with the rise of social media over the past decade. Moreover *information networks* such as hyperlink structure of the Web and blog networks [Leskovec et al., 2009], have already proven their effectiveness in a plethora of applications to ease the navigation through these structures for instance. Among this set of applications of graph analysis, which even extends to data that does not inherently contain any graph structure, consider for example, text data transformation to graph of terms, where term proximity(ies) represents relationship(s) in the graph [Rousseau and Vazirgiannis, 2013]. *Biological* applications also contribute to a large part of graph analysis. Indeed, in neuroscience, where precise relations such as protein-protein interactions, or for seizure detection [Wang et al., 2017], graph representations are omnipresent. In a more straightforward way, graphs have been widely used as molecule modeling for biological compound classification

and brain network understanding [de Vico Fallani et al., 2014]. The later known to have inspired the design of the first *multilayer perceptron* [LeCun et al., 2015].

The aforementioned network designs and domains of application call for structures that are not governed by randomness. Indeed *social networks*, as well as protein-protein interaction graphs [Faloutsos et al., 1999], tend to follow a power-law degree distribution. The underlying inhomogeneity of the edge distribution needs for denser parts in the graph and a variance in the connectivity within the graph that highlights meaningful substructures.

Network Science has already tackled efficiently many of the above problems and took interest in all of those applications, providing efficient algorithms for data that requires all the more computational power. These algorithms come usually either from random graph theory such as Markov chains, for example the pagerank algorithm [Page et al., 1999], or from graph combinatorics which will be the main focus of our work. In this sense, the k-core decomposition, introduced in [Seidman, 1983a] and inherited from the latter field, is proved to be a major tool in several graph mining tasks.

Most of the research conducted on the degeneracy concepts proved to provide state-of-the art frameworks. Indeed, k-truss decomposition, introduced in [Cohen, 2008, Wang and Cheng, 2012], and D-core decomposition, introduced in [Giatsidis et al., 2011], are based on triangle cohersiveness for the former and on an adaptation to directed graphs of the k-core decomposition for the latter. Both of these structural decompositions became milestones in their given fields of applications, namely for community detection, finding influential spreaders and measuring collaboration quality. It is important, though, to highlight that all these frameworks capitalize on the vertex degrees of the graph, meaning that we only look at the vertex degree density.

Moreover, a natural assumption will be to look at edge degree of the nodes to provide a degeneracy type decomposition. As our aim is, at first, to try to unify degeneracy frameworks not only by applying it to new types of graphs, which will still be some of our concern, but as well to learn structural properties

that have for the edge and connectivity based counterparts of the degeneracy.

1.1 Thesis Statement and Research Drives

In this dissertation we will provide new graph decompositions, models and tools for several domains of application, in the twofold scope of both unsupervised and supervised learning. We will build those models in order to :

— Analyze edge degree-based graph decomposition in undirected graphs with possible parallel edges. Also to attempt to find similar theoretical properties that exist for the vertex degree degeneracy analogues.

— Next we try to unify and order the degree degeneracy hierarchies, connectivity degeneracy hierarchies and their admissibility analogues. We also investigate the possible use of an edge-connectivity degeneracy for traditional graph degeneracy related tasks, as described previously.

— In the last two chapters we first manage to apply degeneracy framework to classic supervised learning frameworks, namely graph kernels and neural networks, and achieve state-of-the-art performance as well as designing a degeneracy framework adapted to hypergraphs and bipartite graphs.

An independent branch of research on graph searching is the Cops and Robber games, defined in [Nowakowski and Winkler, 1983] and since then, many versions of this game have been proposed. This game consists in cops and robbers being two entities moving under certain constraints over a graph, where the cops try to block the robber, under given conditions depending on the game setting. As stated previously and at least for the first part of the thesis, our work is based on the k-core decomposition and more precisely on its edge counterpart. Indeed, we first study the edge degeneracy in an adapted version of cops and robbers game. As an independent branch of research on the graph searching domain, Cops and Robber games, defined in [Nowakowski and Winkler, 1983], exist in many different variations. In general this game consists in cops and robbers being two entities moving under certain constraints over a graph, where the cops try to block the robber, under given conditions depending on the

game setting. Under our perspective and version for this game, our study subsequently leads to some structural results, both on the edge degeneracy and on algorithm complexity to find it. More importantly, we prove an edge analogue theorem to Dvorak's result for degeneracy in [Dvořák, 2012] are proposed.
As a consequence, a unified generator for degeneracy and admissibility hierarchies is proposed. We also provide general ordering properties between degeneracy and admissibility hierarchies. It was first driven by the results of [Richerby and Thilikos, 2011] and then generalized, thanks to previous results to most of the degeneracy and admissibility types.

In this way, the driving force of this dissertation relies on mathematical results, coming from combinatorics as well as statistics. Justifying and supporting both our models and results are a response to well known needs in several Network Science related problems.

Finally, another major interest of our work is the adaptation of degeneracy to more 'exotic' graph structures. As mentioned before, degeneracy frameworks exist for undirected, directed, weighted and unweighted graphs. In Spite of it, a widely used structure, that surprisingly is among the latter recall but has not so far dragged interest or has any adapted decomposition suited, is the bipartite graph. Indeed, bipartite graphs are very present in many real-world applications, in recommendation systems for instance, but lack fast and efficient decomposition algorithms such as the k-core. Though k-core can be applied to a bipartite graph : it is still an undirected graph with the only property of having two separate sets of nodes that do not have intercommunication. Unfortunately, the decomposition will not be meaningful. Considering for instance recommendation systems, we would be interested in a decomposition on only one set of the graph, ranking only customers or only products. Moreover, neural networks are also aggregations of bipartite graphs, hence we are very interested into analysing dynamics of the gradient descent in terms of edge weights. All these observations reinforce and highlight the need to have an adapted degeneracy framework.

1.2 Outline of the Thesis and Contributions

The rest of the dissertation is organized as follows. In Chapter 3 we will study the s-edge-degeneracy, first as a tool for an adapted search game of cops and robbers. This study will help us providing complexity results to compute the s-edge-degeneracy, motivating us to try to implement this framework in Chapter 4. In the second part of Chapter 3 we will prove, among other results, a structural theorem for graphs with bounded ∞-edge-admissibility.

Next, in Chapter 4, we will first expand the results of [Richerby and Thilikos, 2011], on degree degeneracy and admissibility hierarchies, to connectivity degeneracy and admissibility for their vertex and edge versions. This could be mostly achieved thanks to the results obtained in Chapter 3. In the second part of this chapter we will showcase applications of the 1-edge-connectivity degeneracy, namely for the application of k-edge-connectivity cores, and design an efficient algorithm to compute it. Unfortunately, this decomposition is not relevant enough, for a variety of reasons that will be developed when applied without refining to real world datasets. On the other hand, the edge-connectivity core decomposition in most of the studied cases is very similar, if not identical, to the one produced with the k-core.

Capitalizing on the latter observation, k-core in real-world datasets has very good connectivity properties. Hence, the rest of the dissertation focuses on using the k-core decomposition in supervised learning framework, knowing that it is not interesting to use more complex decompositions due to the computational complexity trade-off.

In Chapter 5, we use k-core decomposition to refine graph kernel outputs by comparing cores of graphs and by summing the resulting kernels. We also prove that we can predict if our framework can outperform the original kernel used on the whole graphs without the k-core decomposition. As our framework outperforms significantly state-of-the-art kernels, we are all the more motivated to generalize degeneracy frameworks to further advanced architectures, *i.e.* neural networks.

Thus, in the last chapter and before concluding this dissertation, we propose in

Chapter 6, a k-core version for bipartite graphs. This specific design is built to suit neural network architectures, considered as containing blocks of bipartite graphs, *i.e.* pairs of layers convolutional or not. This decomposition is then used to reinitialize the weights of the network after a small period of pretraining, in order to adapt the weights to the corrections, given by the learning process. This method is meant to be used in any architecture as it can be applied to any pair of layers contained in it. In the end, we manage to outperform the standard initialization method, as well as having a faster convergence during the gradient descent.

Finally, we will conclude by detailing our contributions and sharing insights on perspectives, as the two last parts of the thesis open many directions for very exciting future research.

CHAPITRE 2

Basic Notations and Definitions

In this chapter, we provide the basic notations, definitions and material that will be used through this dissertation. We will provide notation details for the different types of graphs that will be mentioned and some main concepts that will frequently appear. A particular attention will be given to the degeneracy framework and definitions that constitutes the central component of the dissertation. In the next chapters we will add definitions needed for the given chapter.

2.1 Graphs, Notations and Types

A graph, or network is a representation of entities with some relationships. These relationships can differ a lot depending on a domain of application and this will be described more specifically with the presentation of the datasets used in the different chapters for model evaluation. As a general convention of notation a graph G is defined by its set of nodes $V(G)$, and edges $E(G)$. Moreover, the number of nodes in a graph is equal to $n = |V(G)|$ and the number of edges is $m = |E(G)|$. Also a graph can be directed, undirected, bipartite, multipartite, can have weighted or unweighted edges, and nodes can have labels or attributes. We will also work in the last chapter with hypergraphs where the hyperedges can contain several nodes.

Definition 2.1.1 (Directed and Undirected Graph)**.** In a directed graph $G_D = (V, E)$, every edge $(i, j) \in E$ links node i to node j (ordered pair of nodes). An undirected graph is a directed graph where, if $(i, j) \in E$ then $(j, i) \in E$.

Definition 2.1.2 (Bipartite Graph)**.** A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets U and V such that every edge connects a vertex in U to one in V. Edges and nodes of a bipartite graph can also have weights and/or attributes.

Definition 2.1.3 (Hypergraph)**.** A hypergraph is a generalization of graph in which an edge can join any number of vertices. It can be represented as $\mathcal{H} = (V, E_{\mathcal{H}})$ where V is the set of nodes, and $E_{\mathcal{H}}$ is the set of hyperedges, i.e. a set of subsets of V. Therefore $E_{\mathcal{H}}$ is a subset of $\mathcal{P}(V)$.

Definition 2.1.4 (Incidence graph)**.** An incidence graph is a bipartite graph where one set of the bipartite graph represents the edges of the initial graph, and the other independent set of nodes of the bipartite graph represents the nodes of the graphs. A link exists in this incidence graph if and only if a node belongs to an edge within the original graph.

Definition 2.1.5 (Path)**.** A path is defined as a sequence of nodes $v_1, v_2, \ldots, v_k$, with the property that every consecutive pair of nodes v_i, v_{i+1} in the sequence is connected by an edge.

Definition 2.1.6 (Tree)**.** A tree is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph.

Definition 2.1.7 (Complete Graph)**.** A graph $G = (V, E)$ is called complete, if every pair of distinct nodes is connected by a unique edge. The complete graph of n nodes has $m = \binom{n}{2}$ edges.

Definition 2.1.8 (Clique)**.** A clique is defined as a subset of the nodes of a graph, such that its induced subgraph is complete. A k-clique is a clique of size k.

Definition 2.1.9 (θ_k graph)**.** The θ_k graph is a graph with two nodes and k parallel edges.

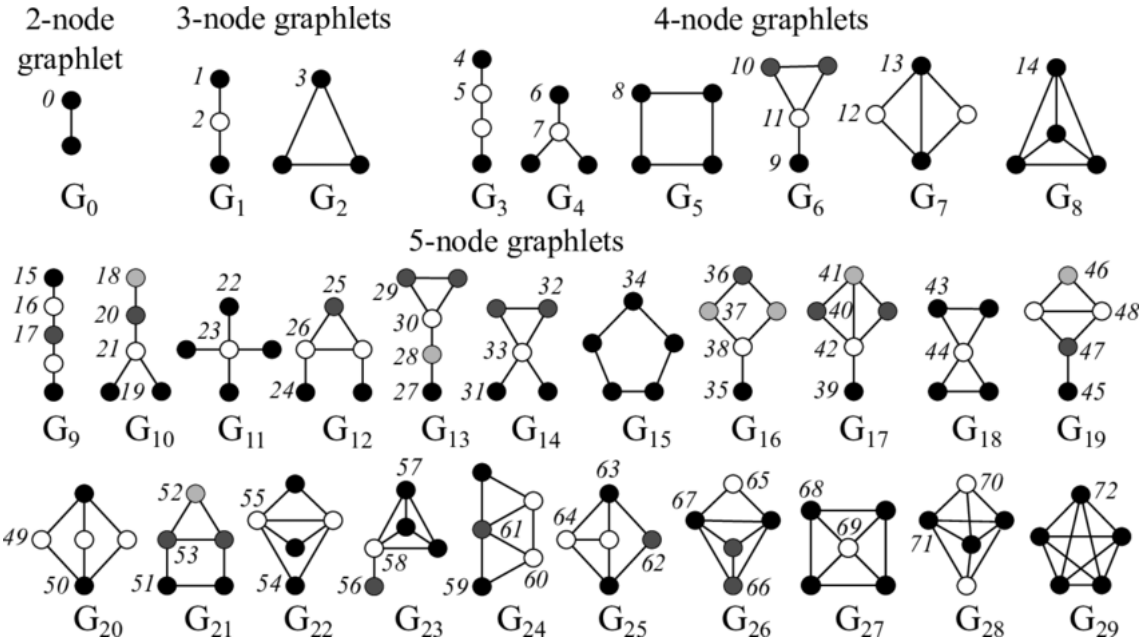


FIGURE 2.1: All graphlets and their automorphism orbits of size 2 to 5. Moreover G_2, G_8, G_29 are also cliques of size 3, 4 and 5. G_0 is also a θ_1-graph

Definition 2.1.10 (Graphlet)**.** A graphlet is a denomination for a small subgraph of a large network parametrized by its size. A network can contain many graphlets of different sizes (see Figure 2.1).

2.2 Graph Functions and Properties

In this section we provide definitions of functions and concepts that will be used and mentioned throughout this dissertation.

Definition 2.2.1 (Connectedness)**.** Let $G = (V, E)$ be an undirected graph two nodes $i, j \in V$ are called connected if there is a path in G from node i to node j.

Definition 2.2.2 (Induced Subgraph)**.** Let $G = (V, E)$ be any graph, and let $S \subseteq V$ be any subset of vertices of G. Then the induced subgraph $G[S]$ is the graph whose vertex set is S and whose edge set consists of all of the edges in E that have both endpoints in S.

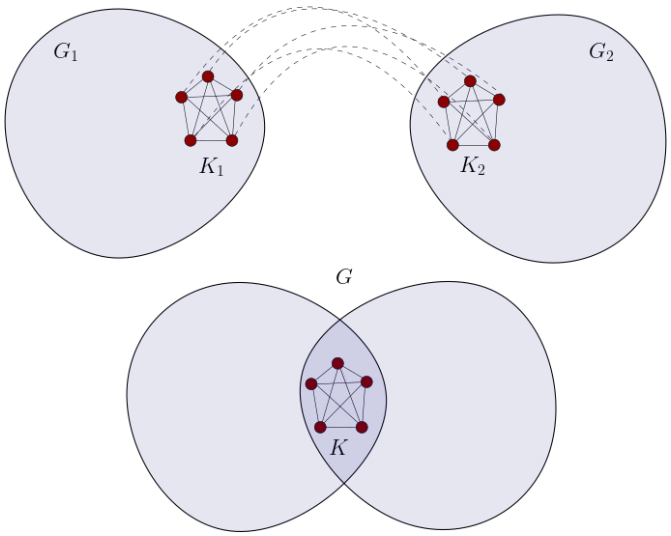


FIGURE 2.2: 5-clique sum of graphs G_1 and G_2.

Definition 2.2.3 (submodular function [Schrijver, 2003])**.** If Ω is a finite set, a submodular function is a set function $f : 2^{\Omega} \to \mathbb{R}$, where 2^{Ω} denotes the power set of Ω , which satisfies one of the following equivalent conditions.

— For every $X, Y \subseteq \Omega$ with $X \subseteq Y$ and every $x \in \Omega \setminus Y$ we have that $f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$.

— For every $S, T \subseteq \Omega$ we have that $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$.

— For every $X \subseteq \Omega$ and $x_1, x_2 \in \Omega \backslash X$ such that $x_1 \neq x_2$ we have that $f(X \cup \{x_1\}) + f(X \cup \{x_2\}) \geq f(X \cup \{x_1, x_2\}) + f(X)$.

Definition 2.2.4 (Clique-Sum Figure 2.2)**.** Let G_1 and G_2 be two graphs with two cliques $K_1 \subseteq V(G_1)$ and $K_2 \subseteq V(G_2)$ of the same size. Graph G is a clique-sum of G_1 and G_2 if it can be obtained by identifying K_1 and K_2, and then removing some of the edges of the clique.

2.2.1 Degeneracy and k-core Decomposition

The k-core decomposition of graphs is a powerful tool for network analysis and it is commonly used as a measure of importance and well connectedness for vertices in a broad spectrum of applications. The study of k-core decomposition and degeneracy goes back to the 60s. More specifically, the first definition

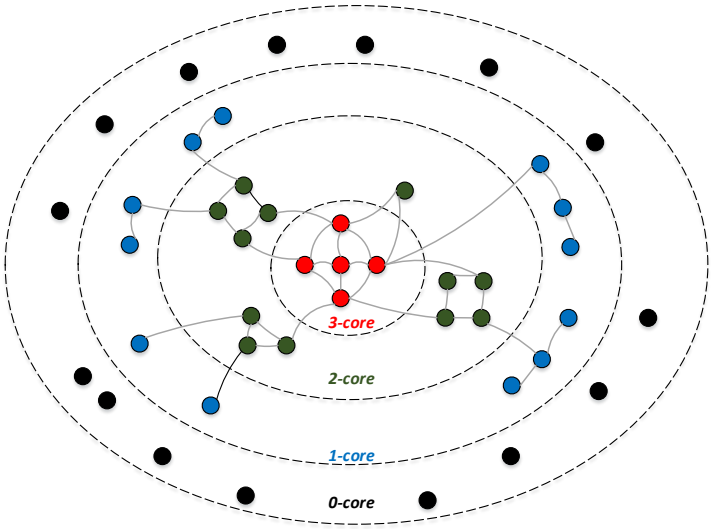


FIGURE 2.3: Example of core decomposition of graph.

of a concept related to k-core (coloring number) was given by Erdős and Hajnal [Erdős and Hajnal, 1966]. The degeneracy of a graph was later defined by Lick and White [Lick and White, 1970]. The notion of k-core was first introduced by Seidman [Seidman, 1983a] to study the cohesion of social networks. In recent years, the k-core decomposition has been established as a standard tool in many application domains such as in network visualization [Alvarez-Hamelin et al., 2006], in protein function prediction [Wuchty and Almaas, 2005] and in graph clustering [Giatsidis et al., 2014]. More formally, let G be a graph and G' a subgraph of G induced by a set of vertices S. Then, G' is defined to be a k-core of G, denoted by C_k, if it is a maximal subgraph of G in which all vertices have degree at least k. Hence, if G' is a k-core of G, then $\forall v \in S$, $d_{G'}(v) \geq k$. Each k-core is a unique subgraph of G, and it is not necessarily connected. The core number $c(v)$ of a vertex v is equal to the highest-order core that v belongs to. In other words, v has core number $c(v) = k$, if it belongs to a k-core but not to any $(k+1)$-core. The degeneracy $\delta^*(G)$ of a graph G is defined as the maximum k for which graph G contains a non-empty k-core subgraph, $\delta^*(G) = \max_{v \in V} c(v)$. Furthermore, assuming that $\mathcal{C} = \{C_0, C_1, \ldots, C_{\delta^*(G)}\}$ is the set of all k-cores, then $\mathcal{C}$ forms a nested chain :

$$C_{\delta^*(G)} \subseteq \ldots \subseteq C_1 \subseteq C_0 = G$$

Since the k-cores of a graph form a nested chain of subgraphs, the k-core decomposition is a very useful tool for discovering the hierarchical structure of graphs. Figure 2.3 depicts an example of a graph and its corresponding k-core decomposition. As we observe, the degeneracy of this graph is $\delta^*(G) = 3$; thus, the decomposition creates four nested k-core subgraphs, with the 3-core being the maximal one. The nested structure of the k-core subgraphs is indicated by the dashed lines. Furthermore, the color on the nodes indicates the core number c of each vertex.

The popularity of the k-core decomposition stems mainly from the fact that it can be computed in linear time [Matula and Beck, 1983, Batagelj and Zaveršnik, 2011]. The algorithm for performing the k-core decomposition of a graph is illustrated in Algorithm 2. The algorithm runs in $\mathcal{O}(n+m)$ time. The underlying idea

Algorithm 1 k-core Decomposition

```
1:  procedure k-CORE(G)
2:      Input : A graph G = (V, E)
3:      Output : A set of k-cores C
4:
5:      C = {V}
6:      k = min_{v∈V} d(v)
7:      for i = 1 to n do
8:          Let v be the vertex with the smallest degree in G
9:          if d(v) > k then
10:             add V to C
11:             k = d(v)
12:         end if
13:         V = V \ {v}
14:     end for
15: end procedure
```

is that we can obtain the i-core of a graph if we recursively remove all vertices with degree less than i and their incident edges from the graph until no other vertex can be removed. Since higher-order cores are nested within lower-order cores, we compute k-cores sequentially from $k = 0$ to $k = \delta^*(G)$. Therefore, at each iteration, the algorithm removes the lowest degree vertex and sets its core number accordingly.

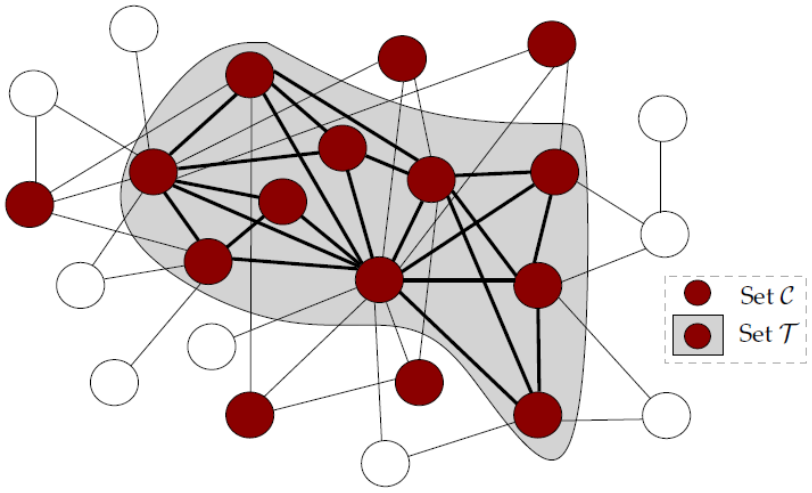


FIGURE 2.4: The set $\mathcal{C}$ represents the top k-core, versus the $\mathcal{T}$ set representing the top k-truss

2.2.2 Extensions of the Decomposition

In the literature there is a research effort to introduce new graph decompositions inspired by the degeneracy framework. This is as well one of our main concerns throughout the whole thesis. One of those decompositions that is worth mentioning is the k-truss decomposition. It is in fact the k-core decomposition of the line graph studied, i.e a k-truss in a graph is a subset of the graph such that every edge in the subject is supported by at least k_2 other edges that form triangles with that particular edge Figure 2.4. In other words, every edge in the truss must be part of k_2 triangles made up of nodes that are part of the truss.

2.3 Comments and Precisions

The previously defined concepts and objects will be widely used, at least for a majority of them, in the rest of the dissertation. It is important to note that we will work exclusively on undirected graphs, that will have some given properties depending on the application. As shown in the previous section, k-core as it is defined is applied to undirected graphs, but can also work in the same manner on edge weighted graphs, where nodes have weighted degree. Moreover, for bipartite graphs, k-core decomposition is known to work poorly as we

might need a core decomposition on only one side of the bipartite graph (this will be extensively studied in Chapter 6). Finally, except for Chapter 5, where k-core decomposition is used as a tool, capitalizing on the properties we learnt in the previous sections. The focus of our work will be, in Chapter 3, to study and explain theoretical properties on the degeneracy and generalised concepts of degeneracy, namely the s-degeneracy. This motivates then our main goal, that is building equivalent theoretical results, and unify degree degeneracy and connectivity degeneracy hierarchies under a single framework that ease the generation of such decompositions. We then experiment on one of these decompositions, namely the 1-edge-connectivity degeneracy, comparing it to the k-core and k-truss decompositions, on real-world scenarios (studied in Chapter 4).

It is important, to avoid any further confusion, to distinguish s-degeneracy and k-core, i.e. the k-core framework corresponds to the 1-degeneracy. This will be extensively developed in Chapter 3 and after this chapter we will only deal with k-core, k-truss, or the later defined k-edge-connectivity core.

CHAPITRE 3

Edge Degeneracy : Algorithmic and Structural Results

After introducing the graph concepts and definitions revolving around degeneracy, specifically the k-core and k-truss, we will develop in this chapter the edge equivalents, namely the s-edge-degeneracy and admissibility. In order to add an applicative context and justify the need for edge alternatives of degeneracy we will introduce an adapted search game and try to provide search strategies based on these edge degeneracy hierarchies.

We consider a cops and robber game where the cops are blocking edges of a graph, while the robber occupies its vertices. At each round of the game, the cops choose some set of edges to block and right after the robber is obliged to move to another vertex traversing at most s unblocked edges (s can be seen as the speed of the robber). Both parts have complete knowledge of the opponent's moves and the cops win when they occupy all edges incident to the robbers position. We introduce the capture cost on G against a robber of speed s. This defines a hierarchy of invariants, namely $\delta_{\mathrm{e}}^1, \delta_{\mathrm{e}}^2, \ldots, \delta_{\mathrm{e}}^\infty$, where $\delta_{\mathrm{e}}^\infty$ is an edge-analogue of the admissibility graph invariant, namely the *edge-admissibility* of a graph. We prove that the problem asking whether $\delta_{\mathrm{e}}^s(G) \leq k$, is polynomially solvable when $s \in \{1, 2, \infty\}$ while, otherwise, it is NP-complete. Our main result is a structural theorem for graphs of bounded

edge-admissibility. We prove that every graph of edge-admissibility at most k can be constructed using $(\leq k)$-edge-sums, starting from graphs whose all vertices, except possibly from one, have degree at most k. Our structural result is approximately tight in the sense that graphs generated by this construction always have edge-admissibility at most $2k-1$. Our proofs are based on a precise structural characterization of the graphs that do not contain θ_r as an immersion, where θ_r is the graph on two vertices and r parallel edges.

3.1 Introduction

All graphs in this chapter are undirected, finite, loopless, and may have parallel edges. We denote by $V(G)$ the set of vertices of a graph G, while we denote by $E(G)$ the multi-set of its edges. We also use the term *s-path of G* for a path of G that has length at most s.

A *(k,s)-hide out* in a graph G is a subset S of its vertices such that, for each vertex $v \in S$, it is not possible to block all s-paths from v to the rest of S by less than k vertices, different than v. The *s-degeneracy* of a graph G, has been introduced in [Richerby and Thilikos, 2011] as the minimum k for which G contains a (k,s)-hide out. s-degeneracy defines a hierarchy of graph invariants that, when $s=1$, gives the classic invariant of graph degeneracy [Matula, 1968, Bodlaender et al., 2006, Kirousis and Thilikos, 1996] and, when $s=\infty$, gives the parameter of ∞-admissibility that was introduced by Dvořák in [Dvořák, 2013] and studied in [Dvořák, 2012, Kierstead and Trotter, 1991, Nešetřil and de Mendez, 2009, Chen and Schelp, 1993, Nesetril and de Mendez, 2012, Weißauer, 2019, Grohe et al., 2015].

In this chapter we introduce and study the edge analogue of the above hierarchy of graph invariants, namely the *s-edge-degeneracy hierarchy*. The new parameter results from the one of s-degeneracy if we replace (k,s)-hide outs by *(k,s)-edge hide outs* where we ask that, for each vertex v of S, it is not possible to block all s-paths from v to the rest of S by less than k edges. It follows that the value of s-edge-degeneracy may vary considerably than the one of s-degeneracy. For instance, consider the graph θ_k consisting of two vertices and

k parallel edges between them. It is easy to see that, for every positive integer s, the s-degeneracy of θ_k is 2, while it s-edge-degeneracy is k (the two vertices form a (k, s)-edge hideout). In other words, s-edge-degeneracy can be seen as an alternative way to extent the notion of degeneracy using edge separators instead of vertex separators.

In Subsection 3.3.1 we introduce two alternative definitions for s-edge- degeneracy, apart from the one using (k, s)-edge hide outs. The first is in terms of a graph searching game and the second is in terms of graph layouts. Next, we prove a min-max theorem supporting the equivalence of the three definitions. As a consequence of this theorem, we can identify the computational complexity of s-edge-degeneracy : it can be computed in polynomial time when $s \in \{1, 2, \infty\}$, while for all other values of s, deciding whether its value is at most k is an NP-complete problem.

Our next step is to provide a structural theorem for the ∞-edge-degeneracy that, from now on, we call ∞-edge-admissibility. For ∞-degeneracy (also known as ∞-admissibility), Dvořák proved the following structural characterization [Dvořák, 2012, Theorem 6].

Proposition 3.1.1. *For every k, there exist constants d_k, c_k and a_k such that every graph G with ∞-admissibility at most k can be constructed by applying $(\leq c_k)$-clique sums starting from graphs where at most d_k vertices have degree at least a_k.*

In the above proposition the $(\leq k)$-*clique sum* operation receives as input two graphs G_1 and G_2 such that each G_i contains a clique K_i with vertex set $\{v_1^i, \ldots, v_\rho^i\}, \rho \leq k$. The outcome of the operation is the graph occurring if we identify v_j^1 and v_j^2 for $j \in \{1, \ldots, \rho\}$ and then remove some of the edges between the identified vertices. While the constants of Proposition 3.1.1 where not specified in [Dvořák, 2012], an alternative proof was recently given by Weißauer in [Weißauer, 2019] where $d_k = k$, $c_k = k$, and $a_k = 2k(k-1)$.

In Section 3.4 we provide a counterpart of Proposition 3.1.1 for the ∞-*edge-admissibility* that is the following

Theorem 3.1.2. *For every k, every graph G with ∞-*edge-admissibility *at most k can be constructed by applying $(\leq k)$-edge sums starting from graphs where at most one vertex has degree at least $k+1$.*

Observe that Theorem 3.1.2 occurs from Proposition 3.1.1 if we replace ∞-*admissibility* by ∞-*edge-admissibility*, if, instead of clique sums, we consider edge sums, and if we set $d_k = 1$, $c_k = k$, and $a_k = k+1$. The $(\leq k)$-edge sum operation (the definition is postponed to Subsection 3.4.1) was defined in [Wollan, 2015] (see also [Giannopoulou et al., 2015]) and can be seen as the edge-counterpart of clique sums.

The proof of our structural theorem is derived by a *precise* structural characterization of the graphs where each pair of vertices is separated by a cut of size at most k. We prove that these graphs are *exactly* those that can be constructed using $(\leq k)$-edge sums from graphs where all but one of their vertices have degree at most k. This directly implies our structural theorem for ∞-*edge-admissibility*, as every pair of two vertices linked by $k+1$ pairwise edge-disjoint paths is a $(k+1,\infty)$-edge hide out.

Our last result is that the converse of the structural characterization in Theorem 3.1.2 holds in an approximate way : if G can be constructed using $(\leq k)$-edge sums from graphs where all but one of their vertices have degree at most k, then the ∞-edge-admissibility of G is at most $2k-1$. This suggests that our decomposition theorem is indeed the correct choice for the parameter of ∞-edge admissibility.

3.2 Basic Definitions

Sets and integers. Given a non-negative integer s, we denote by $\mathbb{N}_{\geq s}$ the set of all non-negative integers that are not smaller than s. We also denote $\mathbb{N}^+_{\geq s} = \mathbb{N}_{\geq s} \cup \{\infty\}$. Given two integers $p \leq q$, we set $[p,q] = \{p, p+1, \ldots, q\}$ and given a $k \in \mathbb{N}_{\geq 0}$ we define $[k] = [1,k]$. Given a set A, we use 2^A for the set of all its subsets, we define $\binom{A}{2} := \{S \mid S \in 2^A \text{ and } |S| = 2\}$, and, given a $k \in \mathbb{N}_{\geq 0}$ we denote by $A^{(\leq k)}$ the set of all subsets of A that have size at most k. A *near-partition* of a set A is a collection of pairwise disjoint sets whose union is A. A

bipartition of $A, |A| \geq 2$ is a near-partition of A into two non-empty sets.

Graphs. All graphs in this chapter are undirected, finite, loopless, and may have parallel edges. We denote by $V(G)$ the set of vertices of a graph G while we use $E(G)$ for the multi-set of its edges. Given a graph G and a vertex v, we define $E_G(v)$ as the multi-set of all edges of G that are incident to v. We define the *neighborhood* of v as $N_G(v) = (\bigcup_{e \in E_G(v)} e) \setminus \{v\}$, the *edge-degree* of v as $\mathbf{deg}_G(v) = |E_G(v)|$. We also define $\Delta(G) = \max\{\mathbf{deg}_G(v) \mid v \in V(G)\}$. Given a $F \subseteq E(G)$, we define $G \setminus F = (V(G), E(G) \setminus F)$.

Given a tree T and two vertices $a, b \in V(T)$ we define aTb as the path of T connecting a and b. Let G be a graph and let $S_1, S_2 \subseteq V(G)$ where $S_1 \cap S_2 = \emptyset$. We define

$$E_G(S_1, S_2) = \{e \in E(G) \mid e \cap V_1 \neq \emptyset \text{ and } e \cap V_2 \neq \emptyset\}.$$

A *cut* of a graph G is any bipartition $(X, \overline{X})$ of its vertices. The *edges* of a cut $(X, \overline{X})$ is the set $E(X, \overline{X})$ while the *size* of $(X, \overline{X})$ is equal to $|E(X, \overline{X})|$. Given two distinct vertices x and y of G, an (x, y)-*cut* of G is a cut $(X, \overline{X})$ of G such that $x \in X$ and $y \in \overline{X}$.

We define the function $\rho : 2^{V(G)} \to \mathbb{N}$ such that $\rho(X) = |E_G(X, V(G) \setminus X)|$. It is easy to see that ρ is a submodular function, ie.,

$$\forall X, Y \in 2^{V(G)} \quad \rho(X \cap Y) + \rho(X \cup Y) \leq \rho(X) + \rho(Y). \tag{3.1}$$

Given a graph G and two distinct $x, y \in V(G)$, we call an (x, y)-s-*path* every s-path in G starting from x and finishing on y. We also use the term (x, y)-*path* as a shortcut for (x, y)-∞-path. We define the function $\mathbf{cut}_{G,s} : \binom{V(G)}{2} \to \mathbb{N}_{\geq 0}$ so that $\mathbf{cut}_{G,s}(x, y)$ is equal to the minimum size of a $F \subseteq E(G)$ such that $G \setminus F$ does not contain any (x, y)-s-path. The complexity of computing $\mathbf{cut}_{G,s}(x, y)$ is provided by the next proposition (see [Baier et al., 2006, Mahjoub and McCormick, 2010, Itai et al., 1982]).

Proposition 3.2.1. *If $s \in \{1, 2, \infty\}$, then the problem that, given a graph G, a $k \in \mathbb{N}$, and two distinct vertices a and b of G, asks whether $\mathbf{cut}_{G,s}(a, b) \leq k$ is*

polynomially solvable, while it is NP*-complete if* $s \in \mathbb{N}_{\geq 4}$.

3.3 Graph Searching and s-edge-degeneracy

3.3.1 A Search Game

The study of graph searching parameters is an active field of graph theory. Several important graph parameters have their search-game analogues that provide useful insights on their combinatorial and algorithmic properties. (For related surveys, see [Fomin and Thilikos, 2008, Bienstock, 1991, Fomin and Petrov, 1996, Alspach, 2004, Alpern and Gal, 2003].)

We introduce a graph searching game, where the opponents are a group of cops and a robber. In this game, the cops are blocking *edges* of the graph, while the robber resides on the *vertices*. The first move of the game is done by the robber, who chooses a vertex to occupy. Then, the game is played in rounds. In each round, first the cops block a set of edges and next the robber moves to another vertex via a path consisting of at most s unblocked edges. The robber *cannot stay put* and he/she is captured if, after the move of the cops, all the edges incident to his/her current location are blocked. Both cops and robbers have full knowledge of their opponent's current position and they take it into consideration before they make their next move. We next give the formal definition of the game.

The game is parameterized by the speed $s \in \mathbb{N}^+_{\geq 1}$ of the robber. A *search strategy on* G for the cops is a function $f : V(G) \to 2^{E(G)}$ that, given the current position $x \in V(G)$ of the robber in the end of a round, outputs the set $f(v)$ of the edges that should be blocked in the beginning of the next round. The *cost* of a cop strategy f is defined as $\mathbf{cost}(f) = \max\{|f(v)| \mid v \in V(G)\}$, i.e., the maximum number of edges that may be blocked by the robbers according to f.

An *escape strategy on* G for the robber is a pair $R = (v_{\text{start}}, g)$ where v_{start} is the vertex of robber's first move and $g : 2^{E(G)} \times V(G) \to V(G)$ is a function that, given the set F of blocked edges in the beginning of a round and the current position x of the robber, outputs the vertex $u = g(F, v)$ where the robber should move. Here the natural restriction for g is that there is an s-path from v to u in

$G \setminus F$. Clearly, if F is the set of edges that are incident to v, then $g(F, v)$ should be equal to v and this expresses the situation where the robber is captured.

Let f and $R = (v_{\text{start}}, g)$ be strategies for the cop and the robber respectively. The *game scenario* generated by the pair (f, R) is the infinite sequence $v_0, F_1, v_1, F_2, v_2, \ldots$, where $v_0 = v_{\text{start}}$ and for every $i \in \mathbb{N}_{\geq 1}$, $F_i = f(v_{i-1})$ and $v_i = g(F_i, v_{i-1})$. If $v_i = v_{i-1}$ for some $i \in \mathbb{N}_{\geq 1}$, then (f, R) is a *cop-winning* pair, otherwise it is a *robber-winning* pair.

The *capture cost against a robber of speed* s in a graph G, denoted by $\mathbf{cc}_s(G)$ is the minimum k for which there is a cop strategy f, of cost at most k, such that for every robber strategy R, (f, R) is a cop-winning pair.

3.3.2 A Min-max Theorem For s-edge-degeneracy

Definition 3.3.1 (edge-separator and support)**.** Let G be a graph, $x \in V(G)$, $S \subseteq V(G) \setminus \{x\}$, and $s \in \mathbb{N}^+_{\geq 1}$. We say that a set $A \subseteq E(G)$ is an (s, x, S)-*edge-separator* if every s-path of G from x to some vertex in S, contains some edge from A.

We define $\mathbf{supp}_{G,s}(x, S)$ to be the minimum size of an (s, x, S)-edge-separator in G.

Let G be a graph and let $L = \langle v_1, \ldots, v_r \rangle$ be a layout (i.e. linear ordering) of its vertices. Given an $i \in [r]$, we denote $L_{\leq i} = \langle v_1, \ldots, v_i \rangle$. Given an $s \in \mathbb{N}^+_{\geq 1}$, we define the s-*edge-support* of a vertex v_i in L as $\mathbf{supp}_{G,s}(v_i, L_{\leq i-1})$.

Definition 3.3.2 (s-edge-degeneracy)**.** The s-*edge-degeneracy* of L, is the maximum s-edge-support of a vertex in L. The s-*edge-degeneracy* of G, denoted by $\delta^s_{\text{e}}(G)$ is the minimum s-edge-degeneracy over all layouts of G.

Definition 3.3.3 ((k, s)-edge-hide-outs)**.** Let $s \in \mathbb{N}^+_{\geq 1}$ and $k \in \mathbb{N}$. A (k, s)-*edge-hide-out* in a graph G is any set $R \subseteq V(G)$ such that, for every $x \in R$, $\mathbf{supp}_{G,s}(x, R \setminus \{x\}) \geq k$.

A (k, s)-*edge-hide-out* S is *maximal* if there is no other (k, s)-*edge-hide-out* S' with $S \subsetneq S'$. It is easy to verify that every graph contains a unique maximal (k, s)-edge-hide-out.

(k, s)-edge-hide-outs can be seen as obstructions to small s-edge-degeneracy. In particular we prove the following min-max theorem, characterizing the search game that we defined in Subsection 3.3.1.

Theorem 3.3.4. *Let G be a graph and let $s \in \mathbb{N}^+_{\geq 1}$ and $k \in \mathbb{N}$. The following three statements are equivalent.*

(1) $\mathbf{cc}_s(G) \leq k$, i.e., there is a cop strategy f on G of cost less than k, such that for every robber strategy R on G, (f, R) is cop-winning.

(2) G has no $(k+1, s)$-edge-hide-out.

(3) $\delta^s_{\mathrm{e}}(G) \leq k$.

Démonstration. *(1) $\Rightarrow$ (2)*. We prove that the negation of *(2)* implies the negation of *(1)*. Suppose that S is a $(k+1, s)$-edge-hide-out of G. We use S in order to build an escape strategy $R = (v_{\mathrm{start}}, g)$ on G as follows : Let v_{start} be any vertex in S. Let now $v \in S$ and $F \in 2^{E(G)}$. If $|F| > k$, then $g(v, F) = v$. We next define $g(v, F)$ for every $F \in E(G)^{\leq k}$. As S is a $(k+1, s)$-edge-hide-out of G, we know that $\mathbf{supp}_{G,s}(v, S \setminus \{v\}) \geq k+1$, therefore there is an s-path from v to some vertex $u \in S \setminus \{v\}$ that avoids all edges in F. We define $g(v, F) = u$. Notice now that if f is a cop strategy on G of cost at most k, and $v_0, F_1, v_1, F_2, v_2, \ldots$, is the game scenario generated by the pair (f, R), then $v_{i-1} \neq v_i$ for every $i \in \mathbb{N}_{\geq 1}$. This means that R is a robber-winning strategy against any cop strategy of cost at most k, therefore $\mathbf{cc}_s(G) \geq k + 1$.

(2) $\Rightarrow$ (3). Let $n = |V(G)|$. As G has no $(k+1, s)$-edge-hide-out, it follows that for every $R \subseteq V(G)$ there is a vertex $v \in R$, such that $\mathbf{supp}_{G,s}(v, R \setminus \{v\}) \leq k$. We pick such a vertex for every $R \subseteq V(G)$ and we denote it by $v(R)$. We now set $V_n = V(G)$, $v_n = v(V_n)$, and for $i \in \langle n-1, \ldots, 1\rangle$ we set $V_i = V_{i+1} \setminus \{v_{i+1}\}$, $v_i = v(V_i)$. We now set $L = \langle v_1, \ldots, v_n\rangle$ and observe that for every $i \in [n]$, $\mathbf{supp}_{G,s}(v_i, L_{\leq i-1}) = \mathbf{supp}_{G,s}(v_i, V_{i-1}) \leq k$. Therefore, the s-edge-degeneracy of L is at most k, hence $\delta^s_{\mathrm{e}}(G) \leq k$.

(3) $\Rightarrow$ (1). Suppose now that $L = \langle v_1, \ldots, v_n\rangle$ is a layout of $V(G)$ such that, for every $i \in [n]$, $\mathbf{supp}_{G,s}(v_i, L_{\leq i-1}) \leq k$. We use L to build a cop strategy $f : V(G) \to 2^{E(G)}$ as follows. Let $i \in [n]$ and let F_i be an $(s, v_i, L_{\leq i-1})$-edge-separator of G. We define f by setting $f(v_i) = F_i$. This means that if at some

point the robber occupies vertex v_i, then there is no s-path in $G \setminus F_i$ from v_i to $L_{\leq i-1}$. As a consequence of this, no matter what the robber strategy $R = (v_{\text{start}}, g)$ is, it should hold that $g(v_i, F_i) \in L_{\geq i}$. Therefore if $x_0, F_1, x_1, F_2, x_2, \ldots,$ is the game scenario generated by the pair (f, R), then $x_i = x_{i-1}$ for some $i < n$. □

3.3.3 The Complexity of s-edge-degeneracy, for Distinct Values of s

We now combine Proposition 3.2.1 with the min-max theorem of the previous subsection in order to identify the computational complexity of δ_{e}^s for different values of s. Our main result is the following.

Theorem 3.3.5. *If $s \in \{1, 2, \infty\}$, then the problem that, given a graph G and a $k \in \mathbb{N}$, asks whether $\delta_{\text{e}}^s(G) \leq k$, is polynomially solvable, while it is* NP*-complete if $s \in \mathbb{N}_{\geq 4}$.*

Démonstration. Notice first that checking whether $\delta_{\text{e}}^s(G) \leq k$ can be done by the algorithm **check s-edge degeneracy** in 3. Indeed, if the maximal $(k+1, s)$-edge-hideout S is non-empty then the above algorithm will report that $\delta_{\text{e}}^s(G) > k$ after visiting, in line 3, every vertex not in S, as, by the maximality of S, for every $S' \supsetneq S$ there is a vertex $x \in S' \setminus S$ where $\mathbf{supp}_{G,s}(x, S' \setminus \{x\}) \leq k$. On the other hand, if S is empty, then the procedure will produce a layout $L = \langle v_1, \ldots, v_n \rangle$ with s-edge-degeneracy at most k.

Clearly, **check s-edge degeneracy** runs in polynomial time if checking whether $\mathbf{supp}_{G,s}(x, S \setminus \{x\}) \leq k$ can be done in polynomial time, which is equivalent to checking whether $\mathbf{cut}_{G',s}(x, x') \leq k$ where G' is the graph obtained by G after we identify all vertices of $S \setminus \{x\}$ to a single vertex x'. As this is possible for $s \in \{1, 2, \infty\}$, due to Proposition 3.2.1, the polynomial part of the theorem follow.

It now remains to prove that checking whether $\delta_{\text{e}}^s(G) \leq k$ is an NP-hard problem when $s \in \mathbb{N}_{\geq 4}$. For this we will reduce the problem of checking whether $\mathbf{cut}_{G,s}(a, b) \leq k$ to the problem of checking whether $\delta_{\text{e}}^s(G) \leq k$ and the result will follow from the hardness part of Proposition 3.2.1.

Let $\mathbf{T}_s = (G, a, b, k)$ be a quadruple where G is a graph on n vertices, $k \in \mathbb{N}_{\geq 0}$, and a, b two distinct vertices of G. We construct the graph $G_{\mathbf{T}_s}$ as follows :

Algorithm 3 An algorithm checking whether $\delta_{\mathrm{e}}^{s}(G) \leq k$.

procedure CHECK EDGE-DEGENERACY(G,k)
  Input : a graph G and an integer $k \in \mathbb{N}_{\geq 0}$
  Output : a report on whether $\delta_{\mathrm{e}}^{s}(G) \leq k$

  $n \leftarrow |V(G)|$, $S \leftarrow V(G)$
  for $i = n$ **to** 1 **do**
    if there is an $x \in S$ with $\mathbf{supp}_{G,s}(x, S \setminus \{x\}) \leq k$ **then**
      $v_i \leftarrow x$
    else
      report that "$\delta_{\mathrm{e}}^{s}(G) > k$" and **stop**
                                  ▷ S is the maximal $(k+1, s)$-edge-hideout of G,
                                  ▷ witnessing that $\delta_{\mathrm{e}}^{s}(G) > k$, because of Theorem 3.3.4.
    end if
    $V = V \setminus \{v\}$
  end for
  Output "$\delta_{\mathrm{e}}^{s}(G) \leq k$, witnessed by layout $L = \langle v_1, \ldots, v_n \rangle$.
end procedure

Take $k+n+1$ copies $G_1, \ldots, G_{k+n+1}$ of G and identify all a's of these copies to a single vertex that we call again a, while we set $B := \{b_1, \ldots, b_{k+n+1}\}$ where b_i is the copy of b in G_i. Next, we add n new vertices $C = \{c_1, \ldots, c_n\}$ and, for every $(i,j) \in [n] \times [k+n+1]$, we add the edge $e_{i,j} = c_i b_j$. The construction of $G_{\mathbf{T}_s}$ is completed by subdividing each edge $e_{i,j}$ $s-1$ times.

For every $(i,j) \in [n] \times [k+n+1]$, we denote by $P_{i,j}$ the (c_i, b_j)-s-path that replaces $e_{i,j}$ after this subdivision. Also we set

$$\mathcal{P}_j = \{P_{i,j} \mid i \in [n]\}, \text{ for } j \in [k+n+1],$$

$$\mathcal{Q}_i = \{P_{i,j} \mid j \in [k+n+1]\}, \text{ for } i \in [n],$$

and $P = \bigcup_{j \in [k+n+1]} \mathcal{P}_j$.

For the correctness of the reduction, it remains to prove the following.

$$\delta_{\mathrm{e}}^{s}(G_{\mathbf{T}_s}) \leq k+n \iff \mathbf{cut}_{G,s}(a,b) \leq k \tag{3.2}$$

We first claim that, for every $j \in [k+n+1]$,

$$\mathbf{cut}_{G,s}(a,b) = \mathbf{cut}_{G_{\mathbf{T}_s},s}(a, b_j). \tag{3.3}$$

To see (3.3) observe that none of the (b_j, a)-s-paths of $G_{\mathbf{T}_s}$ contains any vertex

outside G_j, therefore $\mathbf{cut}_{G_{\mathbf{T}_s},s}(a,b_j) = \mathbf{cut}_{G_j,s}(a,b_j) = \mathbf{cut}_{G,s}(a,b)$.

We first prove the ($\Rightarrow$) direction of the (3.2). For this we assume that $\mathbf{cut}_{G,s}(a,b) \geq k+1$ and we show that $G_{\mathbf{T}_s}$ contains a $(k+n+1,s)$-edge-hide-out, which, by Theorem 3.3.4, yields $\delta_{\mathrm{e}}^s(G_{\mathbf{T}_s}) \geq k+n+1$. We claim that $S := C \cup B \cup \{a\}$ is a $(k+n+1,s)$-edge-hide-out of $G_{\mathbf{T}_s}$. As $\mathbf{cut}_{G,s}(a,b) \geq k+1 \geq 1$, we know that for each $j \in [k+n+1]$ there is a (b_j,a)-s-path, say R_j, in $G_{\mathbf{T}_s}$ whose internal vertices are not vertices of any path in P. Moreover, every two paths in $\mathcal{R} := \{R_j \mid j \in [k+n+1]\}$ have only one vertex, that is a in common. The fact that $|\mathcal{R}| = k+n+1$ implies that $\mathbf{cut}_{G_{\mathbf{T}_s},s}(a,B) \geq k+n+1$. Therefore, as $\mathbf{cut}_{G_{\mathbf{T}_s},s}(a,S\setminus\{a\}) \geq \mathbf{cut}_{G_{\mathbf{T}_s},s}(a,B)$, we have that

$$\mathbf{cut}_{G_{\mathbf{T}_s},s}(a,S\setminus\{a\}) \geq k+n+1. \tag{3.4}$$

Consider now the vertex b_j, for some $j \in [k+n+1]$, and notice that that $\mathbf{cut}_{G_{\mathbf{T}_s},s}(b_j,W\cup\{a\}) \geq \mathbf{cut}_{G_{\mathbf{T}_s},s}(a,b_j) + |\mathcal{P}_j|$. Combining this with (3.3) and the fact that $|\mathcal{P}_j| = n$, we obtain that $\mathbf{cut}_{G_{\mathbf{T}_s},s}(b_j,W\cup\{a\}) \geq \mathbf{cut}_{G,s}(a,b)+n \geq k+n+1$. As $\mathbf{cut}_{G_{\mathbf{T}_s},s}(b_j,S\setminus\{b_j\}) \geq \mathbf{cut}_{G_{\mathbf{T}_s},s}(b_j,W\cup\{a\})$, we have that

$$\forall j \in [k+n+1] \quad \mathbf{cut}_{G_{\mathbf{T}_s},s}(b_j,S\setminus\{b_j\}) \geq k+n+1. \tag{3.5}$$

Consider now the vertex c_i, for some $i \in [n]$. Notice that $\mathbf{cut}_{G_{\mathbf{T}_s},s}(c_i,B) \geq |\mathcal{Q}_i| = k+n+1$. As $\mathbf{cut}_{G_{\mathbf{T}_s},s}(c_i,S\setminus\{c_i\}) \geq \mathbf{cut}_{G_{\mathbf{T}_s},s}(c_i,B)$ we obtain that

$$\forall i \in [n] \quad \mathbf{cut}_{G_{\mathbf{T}_s},s}(c_i,S\setminus\{c_i\}) \geq k+n+1. \tag{3.6}$$

It now follows from (3.4), (3.5), and (3.6), that S is an $(k+n+1,s)$-edge-hide-out of $G_{\mathbf{T}_s}$, as required.

We now prove the ($\Leftarrow$) direction of (3.2). The assumption that $\mathbf{cut}_{G,s}(a,b) \leq k$ implies that $\mathbf{cut}_{G_{\mathbf{T}_s},s}(a,b_j) \leq k$, because of (3.3). Therefore there is a set F_j of edges in G_i that blocks every (b_j,a)-s-path of $G_{\mathbf{T}_s}$.

Let $L = \langle v_1,\ldots,v_\ell\rangle$ be any layout of the vertices of $G_{\mathbf{T}_s}$ where

$$L_{\leq k+2n+2} = \langle a,c_1,\ldots,c_n,b_1,\ldots,b_{k+n+1}\rangle \tag{3.7}$$

In order to prove that $\delta_{\mathrm{e}}^{s}(G_{\mathbf{T}_s}) \leq k+n$ it suffices to show that, for each $h \in [\ell]$,

$$\mathbf{supp}_{G,s}(v_h, L_{\leq h-1}) \quad \leq \quad k+n. \tag{3.8}$$

Notice that the vertices of $L_{\leq k+2n+2}$ are the vertices of $S = W \cup B \cup \{a\}$. As each other vertex $v \in V(G) \setminus S$, has degree at most $n-1$ in $G_{\mathbf{T}_s}$, we directly have that (3.8) holds when $h \in [k+2n+3, \ell]$. Let now $v_h = b_j$ for some $j \in [k+n+1]$. Let F_j^* be the edges incident to b_j that are edges of the paths in $\mathcal{P}_j$. Observe that $F_j \cup F_J^*$ blocks in $G_{\mathbf{T}_s}$ all the s-paths from $L_{\leq h-1}$ to b_j. As all the edges in $F_j \cup F_J^*$ have some endpoint in $L_{\geq h}$ and $|F_j| + |F_j^*| \leq k+n$, we conclude that (3.8) holds when $h \in [n+2, k+2n+2]$. Let now $v_h = c_i, i \in [n]$. Notice that the distance in $G_{\mathbf{T}_s}$ between c_i and any vertex in $\{a\} \cup (W \setminus \{c_i\})$ is bigger than s, therefore $\mathbf{supp}_{G,s}(v_h, L_{\leq h-1}) \leq |F_j| + |F_j^*| \leq k+n$ and (3.8) holds when $h \in [2, n+1]$. Finally (3.8) holds trivially when $h = 1$. This completes the proof of (3.2), and the theorem follows. □

3.4 A structural Theorem for Edge-admissibility

This section is dedicated to the statement and proof of our structural characterization for $\delta_{\mathrm{e}}^{\infty}$.

3.4.1 Basic Definitions

Edge-admissibility.

Definition 3.4.1 (∞-admissibility)**.** The *∞-admissibility of a graph G is the minimum k for which there exists a layout $L = \langle v_1, \ldots, v_n \rangle$ of $V(G)$ such that for every $i \in [n]$ there are at most k vertex-disjoint, except for v_i, paths from v_i to $L_{\leq i-1}$ in G.*

If in this definition we replace “vertex-disjoint” by “edge-disjoint” (and we obviously drop the exception of v_i) we have an edge analogue of the admissibility invariant that, because of Menger’s theorem is the same invariant as $\delta_{\mathrm{e}}^{\infty}$. This encourages us to alternatively refer to $\delta_{\mathrm{e}}^{\infty}(G)$ as the *∞-edge-admissibility* of the graph G.

The purpose of this section is to give a structural characterization for graphs of bounded edge-admissibility. For this we need first a series of definitions.

Definition 3.4.2 (Immersion)**.** Given a graph G and two incident edges e and f of G (i.e., edges with a common endpoint) the result of *lifting* e *and* f *in* G is the graph obtained from G after removing e and f and then adding the edge formed by the symmetric difference of e and f. We say that a graph H is an *immersion* of a graph G, denoted by $H \leq G$, if a graph isomorphic to H can be obtained from some subgraph of G after a series of liftings of incident edges.

Given a graph H, we define the class of H*-immersion free graphs* as the class of all graphs that do not contain H as an immersion.

Definition 3.4.3 (Edge sums)**.** Let G_1 and G_2 be graphs, let v_1, v_2 be vertices of $V(G_1)$ and $V(G_2)$ respectively such that $k = \mathbf{deg}_G(v_1) = \mathbf{deg}_G(v_2)$, and consider a bijection $\sigma : E_{G_1}(v_1) \to E_{G_2}(v_2)$, where $E_{G_1}(v_1) = \{e_1^i \mid i \in [k]\}$. We define the k*-edge sum of* G_1 *and* G_2 *on* v_1 *and* v_2*, with respect to* σ, as the graph G obtained if we take the disjoint union of G_1 and G_2, identify v_1 with v_2, and then, for each $i \in \{1, \ldots, k\}$, lift e_1^i and $\sigma(e_1^i)$ to a new edge e^i and remove the vertex v_1.

We say that G is a $(\leq k)$*-edge sum of* G_1 *and* G_2 if either G is the disjoint union of G_1 and G_2 or there is some $k' \in [k]$, two vertices v_1 and v_2, and a bijection σ as above such that G is the k'-edge sum of G_1 and G_2 on v_1 and v_2, with respect to σ.

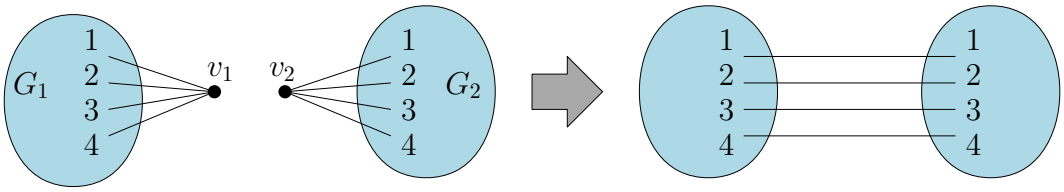


FIGURE 3.1: The graphs G_1 and G_2 and the graph created after the edge-sum of G_1 and G_2.

Let $\mathcal{G}$ be some graph class. We recursively define the $(\leq k)$*-sum closure of* $\mathcal{G}$, denoted by $\mathcal{G}^{(\leq k)}$, as the set of graphs containing every graph $G \in \mathcal{G}$ that is the $(\leq k)$-edge sum of two graphs G_1 and G_2 in $\mathcal{G}$ where $|V(G_1)|, |V(G_2)| < |V(G)|$.

A graph G is *almost k-bounded edge-degree* if all its vertices, except possibly from one, have edge-degree at most k. We denote this class of graphs by $\mathcal{A}_k$.

The rest of this section is devoted to the proof of the the following result.

Theorem 3.4.4. *For every graph G and $k \in \mathbb{N}_{\geq 0}$, if G has edge-admissibility at most k, then G can be constructed by almost k-bounded edge-degree graphs after a series of $(\leq k)$-edge sums, i.e., $G \in \mathcal{A}_k^{(\leq k)}$. Conversely, for every $k \in \mathbb{N}_{\geq 1}$, every graph in $\mathcal{A}_k^{(\leq k)}$ has edge-admissibility at most $2k-1$.*

3.4.2 A Structural Characterizations of θ_k-immersion Free Graphs

Recall that given a $k \in \mathbb{N}_{\geq 1}$, θ_k is the graph with two vertices and k parallel edges between them. In this subsection we prove that θ_k-immersion free graphs are exactly the graphs in $\mathcal{A}_k^{(\leq k)}$ (Theorem 3.4.11).

We need some more definitions in order to translate edge-sums to their decomposition equivalent that will be more easy to handle.

Definition 3.4.5 (Tree-partitions)**.** A *tree-partition* of a graph G is a pair $\mathcal{D} = (T, \mathcal{B})$ where T is a tree and $\mathcal{B} = \{B_t \mid t \in V(T)\}$ is a near-partition of $V(G)$. We refer to the sets in $\mathcal{B}$ as the *bags* of $\mathcal{D}$. Given a tree-partition $\mathcal{D} = (T, \mathcal{B})$ of G and an edge $e \in E(T)$, we define $\mathbf{cross}_{\mathcal{D}}(e) = E_G(V_1, V_2)$, where $V_i = \bigcup_{t \in V(T_i)} B_t$, for $i \in [2]$ and T_1 and T_2 are the two connected components of $T \setminus e$.

For each $t \in V(T)$, we define the t-torso of $\mathcal{D}$ as follows : Let $T_1, \ldots, T_{q_t}$ be the connected components of $T \setminus t$ and let $t_1, \ldots, t_{q_t}$ be the neighbors of t in T such that $t_i \in V(T_i)$. We set $\bar{B}_i = \bigcup_{h \in V(T_i)} B_h$, for $i \in [q_t]$. Next, we define the graph Z_i as the graph obtained from G if, for every $i \in [q_t]$, we identify all the vertices of $\bar{B}_i$ to a single vertex z_i (maintaining the multiple edges created after such an identification). We call Z_t *the t-torso of $\mathcal{D}$* or, simply *a torso* of $\mathcal{D}$. We call the new vertices $z_1, \ldots, z_{q_t}$ *satellites* of the torso Z_t. For each $i \in [q_t]$, we say that z_i *represents* the vertex t_i in T and *subsumes* the connected component T_i of $T \setminus t$. For an example of a tree-partition, see Figure 3.2.

Let $\mathcal{D} = (\mathcal{B}, T)$ be a tree-partition of a graph G. The *adhesion* of $\mathcal{D} = (T, \mathcal{B})$ is $\max\{|\mathbf{cross}_{\mathcal{D}}(e)| \mid e \in E(T)\}$ (the adhesion of the tree-partition of Figure 3.2 is 3). The *strength* of $\mathcal{D} = (T, \mathcal{B})$ is $\min\{\Delta(Z_t) \mid t \in V(T)\}$ (in the tree-partition

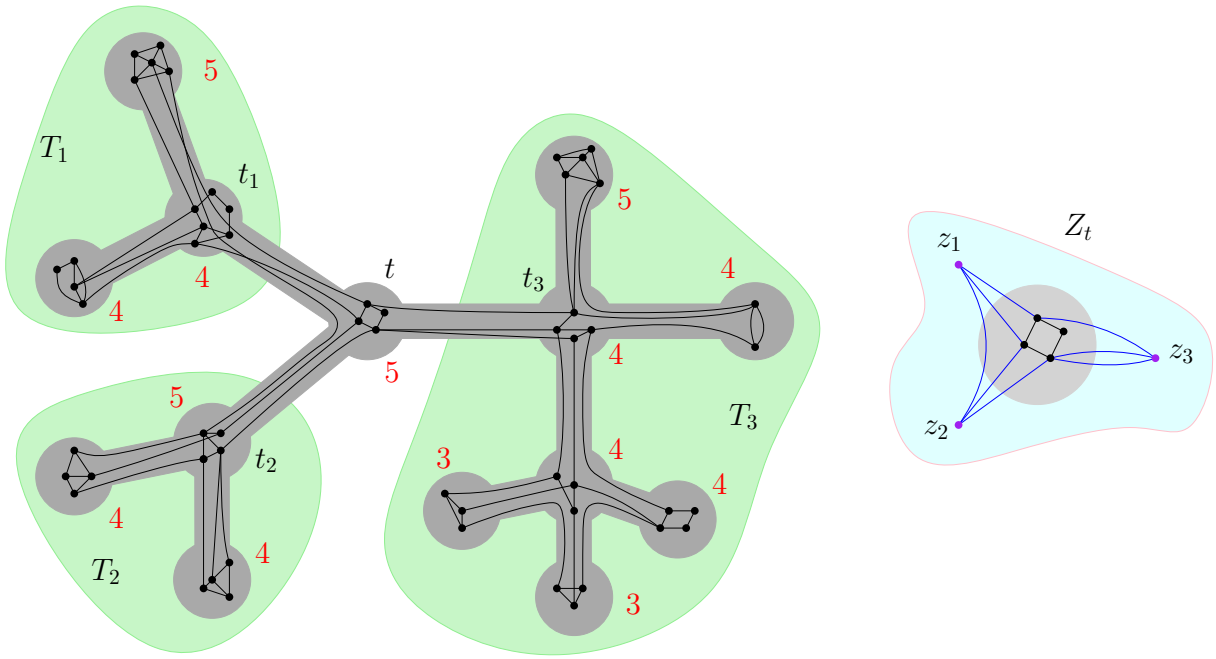


FIGURE 3.2: A graph G, a tree-partition of G with adhesion 3, and the torso Z_t of the vertex t.

of Figure 3.2 the red numbers are the values of $\Delta(Z_t)$ for each node of the tree T).

Observe that if $\mathcal{D}$ has strength at least $k+1$, then every torso of $\mathcal{D}$ contains a vertex of degree at least $k+1$.

Notice that each graph G, where $\Delta(G) \leq k$, has a tree-partition $(T, \mathcal{B})$ where both adhesion and strength are at most k : let T be a star with center r and $|V(G)|$ leaves $\ell_1, \ldots, \ell_{|V(G)|}$, consider a numbering $v_1, \ldots, v_{|V(G)|}$ of $V(G)$, and then set $B_r = \emptyset$, while $B_{\ell_i} = \{v_i\}, i \in [|V(G)|]$.

The next observation follows directly from the definitions and provides a "translation" of edge-sums in terms of tree-partitions.

Observation 3.4.6. Let $\mathcal{G}$ be a graph class and let $k \in \mathbb{N}$. The class $\mathcal{G}^{(\leq k)}$ contains exactly the graphs that have a tree-partition of adhesion at most k whose torsos are graphs in $\mathcal{G}$.

Lemma 3.4.7. *Let $k \in \mathbb{N}_{\geq 0}$ and let G be a graph and $\mathcal{D} = (T, \mathcal{B})$ be a tree-partition of G of adhesion at most k. If $\theta_{k+1} \leq G$, then there is a $t \in V(T)$ such that $\theta_{k+1} \leq Z_t$.*

Démonstration. Observe that if $\theta_{k+1} \leq G$,then there are two vertices x and y in G that are connected by $k+1$ pairwise edge-disjoint (x,y)-paths, $P_1, \ldots, P_{k+1}$ in G. As $\mathcal{D}$ has adhesion at most k, there is some $t \in V(T)$ such that $x, y \in B_t$. Let $T_1, \ldots, T_{q_t}$ be the connected components of $T \setminus t$ and let $z_1, \ldots, z_{q_t}$ be the satellites of the t-torso Z_t of $\mathcal{D}$. Let $i \in [k]$ and notice that, among the edges of the (x,y)-path P_i, those missing from Z_t are those that do not have endpoints in B_t. Notice also that for every $j \in [q_t]$ the edges of P_i with both endpoints in $\bigcup_{t' \in V(T_j)} B_{t'}$ appear as consecutive edges in P_i. We now contract each such set of edges to the vertex z_j for each $j \in [q_t]$ and observe that the resulting path P_i' is a path of Z_t. Observe that $P_1', \ldots, P_{k+1}'$ are pairwise edge-disjoint (x,y)-paths of Z_t and we conclude that $\theta_{k+1} \leq Z_t$ as required. □

Let $\mathcal{D} = (T, \mathcal{B})$ be a tree-partition of a graph G and $k \in \mathbb{N}_{\geq 0}$. We say that a torso Z_t of $\mathcal{D}$ is

— k-*splittable* : if it contains a cut $(X, \overline{X})$ of size smaller than or equal to k where both X and $\overline{X}$ contain some vertex of degree at least $k+1$.

— k-*overloaded* : if at least two of its vertices have degree at least $k+1$.

Given a tree-partition $\mathcal{D} = (T, \mathcal{B})$, we define

$$\mathbf{w}(\mathcal{D}) = \sum_{t \in V(T)} (\mathbf{s}_{\mathcal{D}}(t) - 1)$$

where $\mathbf{s}_{\mathcal{D}}(t)$ is the number of vertices in B_t that have degree at least $k+1$.

Observation 3.4.8. Let G be a graph, $k \in \mathbb{N}_{\geq 0}$, and $\mathcal{D}$ be a tree-partition of G that has strength at least $k+1$. Then $\mathbf{w}(\mathcal{D}) > 0$ iff some of its torsos are k-overloaded.

Given a $k \in \mathbb{N}_{\geq 0}$, we say that a tree-partition $\mathcal{D} = (\mathcal{B}, T)$ is k-*tight* if, its adhesion is at most k and its strength is at least $k+1$.

Lemma 3.4.9. *For every graph G and $k \in \mathbb{N}_{\geq 0}$, if $\mathcal{D}$ is a k-tight tree-partition of G with a k-splittable torso, then there is a k-tight tree-partition $\mathcal{D}'$ of G where $\mathbf{w}(\mathcal{D}') < \mathbf{w}(\mathcal{D})$.*

Démonstration. Let Z_t be a splittable torso of $\mathcal{D}$ and let $L_t = \{z_1, \ldots, z_{q_t}\}$ be the satellite vertices of Z_t. We denote by $t_1, \ldots, t_{q_t}$ be the vertices of T represented

by $z_1, \ldots, z_{q_t}$ respectively. Also we denote by $T_1, \ldots, T_{q_t}$ the connected components of $T \setminus t$ that are subsumed by $z_1, \ldots, z_{q_t}$, respectively. Let also Q_t be the vertices of Z_t that have degree at least $k+1$. As the adhesion of $\mathcal{D}$ is at most k, it follows that each vertex in L_t has degree at most k. Therefore, $Q_t \subseteq B_t$.

We now construct a tree-partition $\mathcal{D}'$ of G. As Z_t is k-splittable, there is a cut $(X, \overline{X})$ of Z_t, of size at most k and two vertices x, y where $\mathbf{deg}_{Z_t}(x), \mathbf{deg}_{Z_t}(y) \geq k+1$, and $x \in X$ and $y \in \overline{X}$. We set $Q_t^{(x)} = Q_t \cap X$ and $Q_t^{(y)} = Q_t \cap \overline{X}$ and keep in mind that $x \in Q_t^{(x)}$ and $y \in Q_t^{(y)}$. Note that there is a set $I \subseteq [q_t]$ such that $X \cap Z_t = \{z_i \mid i \in I\}$ and $\overline{X} \cap Z_t = \{z_i \mid i \in [q_t] \setminus I\}$. We construct the tree T' as follows : we start from $T \setminus t$, then add two new adjacent vertices t_x and t_y, make t_x adjacent with all vertices in $\{t_i \mid i \in I\}$ and make t_y adjacent with all vertices in $\{t_i \mid i \in [q_t] \setminus I\}$. We also define $\mathcal{B}' = \{B'_h \mid h \in V(T')\}$ such that if $h \in V(T) \setminus \{t\}$, then $B'_h = B_h$. Finally, set $B'_{t_x} = B_t \cap X$ and $B'_{t_y} = B_t \cap \overline{X}$. Observe that

— if $e = t_x t_y$, then $|\mathbf{cross}_{\mathcal{D}'}(e)| = \mathbf{cut}_{Z_t}(X, \overline{X}) \leq k$,

— if $e = t_y t_i, i \in [q_t] \setminus I$, then $|\mathbf{cross}_{\mathcal{D}'}(e)| = |\mathbf{cross}_{\mathcal{D}}(tt_i)| \leq k$,

— if $e = t_x t_i, i \in I$, then $|\mathbf{cross}_{\mathcal{D}'}(e)| = |\mathbf{cross}_{\mathcal{D}}(tt_i)| \leq k$, and

— if $e \in E(T') \setminus E(T)|$, then $|\mathbf{cross}_{\mathcal{D}'}(e)| = |\mathbf{cross}_{\mathcal{D}}(e)| \leq k$.

From the above, we deduce that the adhesion of $\mathcal{D}'$ is at most k.

Let now $v \in V(T')$. As $\mathcal{D}$ has strength at least $k+1$, then for each $h \in V(T) \setminus \{t\}$ there is a vertex in B'_h that has degree at least $k+1$. This, together with the fact that $x \in B'_{t_x}$ and $y \in B'_{t_y}$ implies that $\mathcal{D}'$ has strength at least $k+1$. Therefore $\mathcal{D}'$ is k-tight.

We finally observe the following :

— $\mathbf{s}_{\mathcal{D}'}(t_x) = |Q_t^{(x)}|$,

— $\mathbf{s}_{\mathcal{D}'}(t_y) = |Q_t^{(y)}|$, and

— if $t \in V(T') \setminus \{t_x, t_y\}$, then $\mathbf{s}_{\mathcal{D}'}(t) = \mathbf{s}_{\mathcal{D}}(t)$

From the above, $(\mathbf{s}_{\mathcal{D}'}(t_x) - 1) + (\mathbf{s}_{\mathcal{D}'}(t_x) - 1) = |Q_t| - 2 = (\mathbf{s}_{\mathcal{D}}(t) - 1) - 1$, therefore $\mathbf{w}(\mathcal{D}') < \mathbf{w}(\mathcal{D})$ as required. □

Given a tree T and two members a, a' of $E(T) \cup V(T)$ we define aTa' as the unique path in T starting from a and finishing on a'. Also, given a vertex $t \in V(T)$ we define its *status* of t as

$$\mathbf{status}(T, t) = \sum_{t' \in V(T)} |E(tTt')|, \tag{3.9}$$

i.e., the sum of all the lengths of all the paths from t to the rest of the vertices of T.

Let $(X, \overline{X})$ and $(Y, \overline{Y})$ be two cuts of a graph G. We say that the cuts $(X, \overline{X})$ and $(Y, \overline{Y})$ are *parallel* if $X \subseteq Y$, or $\overline{X} \subseteq \overline{Y}$, or $X \subseteq \overline{Y}$, or $\overline{Y} \subseteq X$.

Lemma 3.4.10. *Let $k \in \mathbb{N}_{\geq 0}$. If G is a θ_{k+1}-immersion free graph with at least one vertex of degree at least $k+1$, Then G has a k-tight tree-partition where each torso has exactly one vertex of degree greater than k.*

Démonstration. Notice that G has at least one k-tight tree-partition that consists of a single bag containing all the vertices of G. Among all k-tight tree-partitions of G, consider the set $\mathfrak{D}$ containing every k-tight tree-partition of G, where $\mathbf{w}(\mathcal{D})$ takes the minimum possible value, say ℓ. From Observation 3.4.8 it is enough to prove that $\ell = 0$, i.e., the tree-partitions in $\mathfrak{D}$ contain no k-overloaded torsos. Assume, towards a contradiction, that $\ell > 0$. Consider two vertices x and y, of G each of degree at least $k+1$, that belong to the same bag of some tree-partition of $\mathfrak{D}$. Among all tree-partitions in $\mathcal{D}$ containing x, y in the same bag, say B_t, we choose $\mathcal{D} = (T, \mathcal{B})$ to be one where $\mathbf{status}(T, t)$ is minimized.

As $\theta_{k+1} \not\leq G$, the graph G contains some (x, y)-cut $(X, \overline{X})$ of size at most k. Let $\mathcal{S}_{x,y}$ be the set of all such cuts.

We say that an edge $e \in E(T)$ is *crossed* by $(X, \overline{X})$ if the cut of G corresponding to $\mathbf{cross}_{\mathcal{D}}(e)$ and the cut $(X, \overline{X})$ are not parallel. As both x and y have degree at least $k+1$, there should be two edges e_x and e_y in $\mathbf{cross}_{\mathcal{D}}(e)$ such that $e_x \subseteq X$ and $e_y \subseteq \overline{X}$.

Let $(X, \overline{X}) \in \mathcal{S}_{x,y}$. Let $e = t't''$ be an edge of $E(T)$ that is crossed by $(X, \overline{X})$. We make the convention that, whenever we consider such an edge, we assume that $|E(tTt'')| < |E(tTt')|$, i.e., t'' is closer to t than t', in T. We say that such an edge e is $(X, \overline{X})$-*extremal for* (T, t) if there is no other edge $e' \neq e$ of T that is

crossed by $(X, \overline{X})$ and such that $e \in E(e'Tt)$. We denote by $\mathbf{extr}(X, \overline{X})$ the set of edges of T that are $(X, \overline{X})$-extremal for (T, t). Notice that $\mathbf{extr}(X, \overline{X})$ should be non-empty, as otherwise $(X, \overline{X})$ should induce a cut of Z_t, therefore Z_t would be k-splittable and this, due to Lemma 3.4.9, would contradict the minimality of $\mathbf{w}(\mathcal{D})$. We next define the *cost* of the cut $(X, \overline{X})$ as

$$\mathbf{cost}_{T,t}(X, \overline{X}) = \sum_{t't'' \in \mathbf{extr}(X,\overline{X})} |E(tTt')|.$$

We now pick the (x, y)-cut $(X, \overline{X}) \in \mathcal{S}_{x,y}$ as one of minimum possible cost, in other words, $\mathbf{cost}(X, \overline{X}) = \min\{\mathbf{cost}(X', \overline{X}') \mid (X', \overline{X}') \in \mathcal{S}_{x,y}\}$.

Let $e = t't''$ be an $(X, \overline{X})$-extremal edge of T. Let $(A, \overline{A})$ be the cut of G whose edges are $\mathbf{cross}_{\mathcal{D}}(e)$ and w.l.o.g., we assume that $x, y \in A$. Recall that

$$\rho(X) = \rho(\overline{X}) \leq k \quad \text{and} \quad \rho(A) = \rho(\overline{A}) \leq k. \tag{3.10}$$

We next claim that

$$\rho(A \cap X) = |E(A \cap X, \overline{A} \cup \overline{X})| > k. \tag{3.11}$$

To see (3.11), notice that if this is not the case, then $(A \cap X, \overline{A} \cup \overline{X}) \in \mathcal{S}_{x,y}$, because $x \in A \cap X$ and $y \in \overline{A} \cup \overline{X}$. Notice that if $t = t'$, then $\mathbf{extr}(A \cap X, \overline{A} \cup \overline{X}) = \mathbf{extr}(X, \overline{X}) \setminus \{t''t'\}$ while if t^* is the unique neighbor of t' in the path joining t' and t, then $\mathbf{extr}(A \cap X, \overline{A} \cup \overline{X}) = \mathbf{extr}(X, \overline{X}) \setminus \{t''t'\} \cup \{t't^*\}$. In both cases, $\mathbf{cost}(A \cap X, \overline{A} \cup \overline{X}) = \mathbf{cost}(X, \overline{X}) - 1$, a contradiction to the minimality of the choice of $(X, \overline{X})$. Working symmetrically on $\overline{A}$, instead of A, it follows that

$$\rho(A \cap \overline{X}) = |E(A \cap \overline{X}, \overline{A} \cup X)| > k. \tag{3.12}$$

By the submodularity of ρ, we have that

$$\rho(A \cap X) + \rho(A \cup X) \leq \rho(X) + \rho(A). \tag{3.13}$$

$$\rho(A \cap \overline{X}) + \rho(A \cup \overline{X}) \leq \rho(\overline{X}) + \rho(A). \tag{3.14}$$

Combining now (3.10), (3.11), and (3.13) and (3.10), (3.12), and (3.14) we

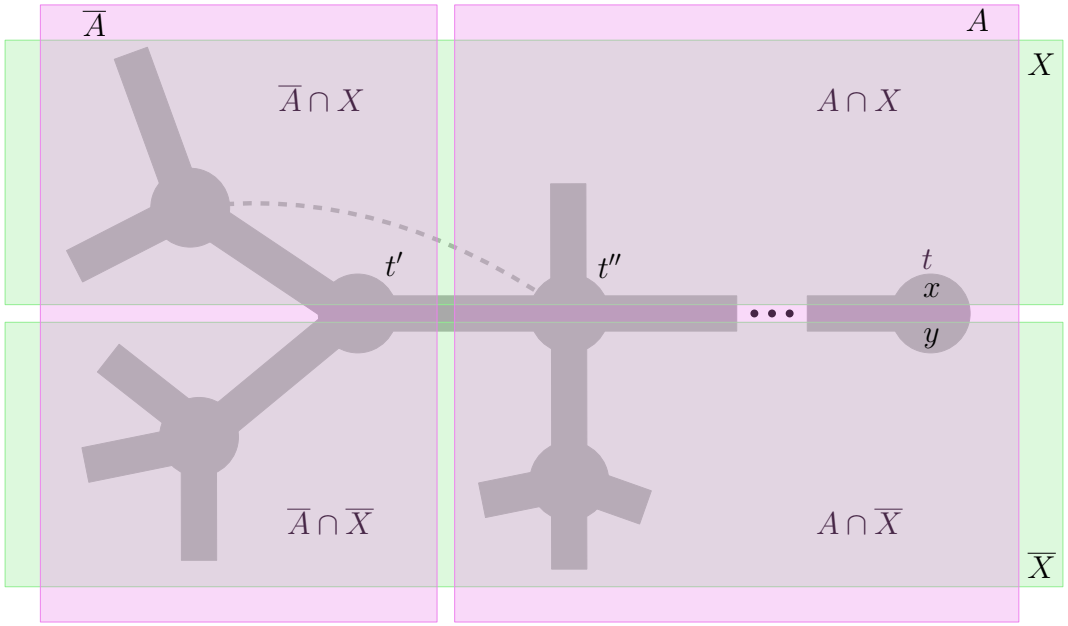


FIGURE 3.3: A visualization of the proof of Lemma 3.4.10.

have that $\rho(A \cup X) \leq k$ and $\rho(A \cup \overline{X}) \leq k$ which can be rewritten

$$\rho(\overline{A} \cap \overline{X}) \leq k \quad \text{and} \quad \rho(\overline{A} \cap X) \leq k. \tag{3.15}$$

Note that the vertices of $B_{t'}$ that have degree at least $k+1$ should all be in exactly one of $\overline{A} \cap \overline{X}$ and $\overline{A} \cap X$. Indeed, if this is not correct, then $Z_{t'}$ should be k-splittable and this, due to Lemma 3.4.9, would contradict the minimality of $\mathbf{w}(\mathcal{D})$. W.l.o.g. we assume that $Q = B_{t'} \cap \overline{A} \cap X$ contains only vertices of degree at most k.

Let $z_1, \ldots, z_{q_{t'}}$ be the satellites of $Z_{t'}$ and let t_i be the vertex of T represented by $z_i, i \in [q_{t'}]$, assuming, w.l.o.g., that z_1 represents t'' in T (that is $t_1 = t''$). Let also T_i be the connected component of $T \setminus t'$ subsumed by z_i, for $i \in [q_{t'}]$. As $t't'' \in \mathbf{extr}(X, \overline{X})$, there is some non-empty $I \subseteq [2, q_{t'}]$ such that

$$\bigcup_{i \in I} \bigcup_{s \in V(T_i)} B_s = (\overline{A} \cap X) \setminus B_{t'} \text{ and } \bigcup_{i \in [2,q_{t'}] \setminus I} \bigcup_{s \in V(T_i)} B_s = (\overline{A} \cap \overline{X}) \setminus B_{t'}. \tag{3.16}$$

We now add the set Q to $B_{t''}$ and remove it from $B_{t'}$, and also remove from T all edges in $\{t_i t' \mid i \in I\}$ and add the edges $\{t_i t'' \mid i \in I\}$ to get T' (in Figure 3.3,

the new edge is depicted by the dashed edge). Observe that $\mathcal{D}' = (T', \mathcal{B})$ is a tree-partition of G with adhesion at most k and where all its nodes contain some vertex of degree at least $k+1$. Therefore $\mathcal{D}'$ is k-tight. Notice that, by the construction of T', $\mathbf{status}(T', t) < \mathbf{status}(T, t)$ a contradiction to the minimality of $\mathbf{status}(T, t)$ in the choice of $\mathcal{D} = (T, \mathcal{B})$. □

Theorem 3.4.11. *For every graph G and $k \in \mathbb{N}$, G is θ_{k+1}-immersion free if and only if $G \in \mathcal{A}_k^{(\leq k)}$.*

Démonstration. We prove first "only if" direction. If G has no vertices of degree at least $k+1$, then $G \in \mathcal{A}_k$ and the result follows trivially. If G has at least one vertex of degree at least $k+1$, then, because of Lemma 3.4.10, G has a k-tight tree-partition of adhesion at most k and whose torsos belong to $\mathcal{A}_k$. Then, from Observation 3.4.6, $G \in \mathcal{A}_k^{(\leq k)}$.

We next prove the "if" direction. Suppose that $G \in \mathcal{A}_k^{(\leq k)}$, therefore, from Observation 3.4.6, G has a tree-partition $\mathcal{D}$ of adhesion at most k whose torsos are all in $\mathcal{A}_k$. As none of the torsos of $\mathcal{D}$ contains θ_{k+1} as an immersion, because of Lemma 3.4.7, the same holds for G and we are done. □

As mentioned by one of the reviewers, Theorem 3.4.11 can alternatively be proved by a suitable application of the theorem of Gomory and Hu [Gomory and Hu, 1961] (see also [Diestel et al., 2019] and [DeVos et al., 2013]).

3.4.3 An Upper Bound to Edge-admissibility

In this subsection we prove that θ_{k+1}-immersion free graphs have edge-admissibility at most $2k-1$. In the end of this section, this will serve for proving Theorem 3.4.4.

Carving decompositions. Given a tree T we denote by $L(T)$ the set of all the vertices of T that have degree at most 1 and we call them the *leaves* of T.

Definition 3.4.12 (Rooted tree and Binary rooted tree)**.** A *rooted tree* is a pair $\mathbf{T} = (T, r)$ where T is a tree and $r \in V(T)$.
A *binary rooted tree* is a rooted tree $\mathbf{T} = (T, r)$ where all its non-leaf vertices have exactly two children.

If $v \in V(T)$, we define $\mathbf{descl_T}(v)$ as the set containing every leave ℓ of T such that $v \in V(rT\ell)$.

Let G be a graph and $S \subseteq V(G)$.

Definition 3.4.13 (Rooted carving decomposition)**.** A *rooted carving decomposition* of G is a pair $(\mathbf{T}, \sigma)$ consisting of a rooted binary tree $\mathbf{T} = (T, r)$ and a function $\sigma : V(G) \to L(T)$.

We stress that σ is not a bijection, i.e., we permit many vertices of G to be mapped to the same leaf of T. The *weight* of a vertex t in $V(T) \setminus L(T)$ is defined as

$$\mathbf{w}(t) = |E_G(S_1, S_2)|$$

where $S_i = \sigma^{-1}(\mathbf{descl_T}(t_i)), i \in [2]$ and t_1, t_2 are the children of t in T. For every edge $e = tt'$ of $E(T)$, where t' is a child of t, we define $\mathbf{cut}(e)$ as the set $E_G(V_1, V_2)$ where $V_1 = \sigma^{-1}(\mathbf{descl_T}(t'))$ and $V_2 = V(G) \setminus V_1$. We also define the *weight* of $e = tt'$ as $\mathbf{w}(e) = |\mathbf{cut}(e)|$.

Lemma 3.4.14. *Let G be a graph and $k \in \mathbb{N}_{\geq 1}$. If $\theta_{k+1} \not\leq G$, then $\delta_{\mathrm{e}}^{\infty}(G) \leq 2k-1$.*

Démonstration. We show that if G is θ_{k+1}-immersion free, then G cannot contain a $(2k, \infty)$-edge-hideout and therefore, from Theorem 3.3.4, $\delta_{\mathrm{e}}^{\infty}(G) \leq 2k - 1$. Suppose to the contrary that $S, |S| \geq 2$, is a $(2k, \infty)$-edge-hideout of G. We build a rooted carving decomposition of G by applying the following procedure :

Step 1. Consider $(\mathbf{T}, \sigma)$ where $\mathbf{T} = (T, v)$, T consists of only one vertex, that is the root r, and $\sigma(v) = r$ for all $v \in V(G)$.

Step 2. Let ℓ be a vertex of T where $|\sigma^{-1}(\ell) \cap S| \geq 2$. If no such vertex exists, then **stop**.

Step 3. Pick, arbitrarily, two distinct vertices x_1 and x_2 in $\sigma^{-1}(\ell) \cap S$. Notice that G contains a (x_1, x_2)-cut (X^1, X^2) of at most k edges where $x_i \in X^i, i \in [2]$, otherwise, from Menger's theorem there are $k + 1$ pairwise edge disjoint paths from x_1 to x_2 in G, which implies the existence of θ_{k+1} as an immersion in G, a contradiction. We now add in T two new vertices ℓ_1 and ℓ_2 make them the children of ℓ and update σ so that the vertices in $X^i \cap \sigma^{-1}(\ell)$ are now mapped in $\ell_i, i \in [2]$, i.e. we remove from σ $(t, \sigma^{-1}(\ell))$ and we add $(t_1, X^1 \cap \sigma^{-1}(\ell))$ and $(t_2, X^2 \cap \sigma^{-1}(\ell))$.

Step 4. Go to **Step 2**.

Let $(\mathbf{T}, \sigma)$ be the rooted carving decomposition produced by the above procedure. By the construction of $(\mathbf{T}, \sigma)$, each vertex of T has weight at most k and for each leaf $\ell \in L(G)$, $|\sigma^{-1}(\ell) \cap S| = 1$. We construct a path P of T by applying the following procedure.

Step 1. Let P be the path of T consisting of r and one (arbitrarily chosen), say t', of the children of r (i.e., P is just an edge). Notice that $\mathbf{w}(\{r, t'\}) = \mathbf{w}(r) \leq k \leq 2k - 1$ (recall that $k \geq 1$).
Step 2. Let e be the the last edge of P (starting from r) and let t be its endpoint that is also an endpoint of P (different than r). If t is a leaf of T, then **stop**.
Step 3. Let t_1 and t_2 be the children of t and let $e_i = tt_i, i \in [2]$. We partition the edges of $\mathbf{cut}(e)$ into two sets, namely F_1 and F_2 so that F_i contains edges with an endpoint in $\mathbf{descl_T}(t_i), i \in [2]$. Notice that $\mathbf{cut}(e_i) = F_i \cup E_G(\sigma^{-1}(\mathbf{descl_T}(t_1)), \sigma^{-1}(\mathbf{descl_T}(t_2)))$, therefore, for $i \in [2]$,

$$\mathbf{w}(e_i) = |\mathbf{cut}(e_i)| \quad = \quad |F_i| + |\mathbf{w}(t)|. \tag{3.17}$$

As $\mathbf{w}(e) \leq 2k - 1$, one, say F_1, of F_1, F_2 should have at most $k - 1$ edges. By applying (3.17) for $i = 1$, we obtain that $|\mathbf{w}(e_1)| \leq k - 1 + \mathbf{w}(t) \leq 2k - 1$. We now extend P by adding in it the vertex t_1 and the edge e_1 and we update $e := e_1$.
Step 4. Go to **Step 2**.

We just constructed a path P in T between r and a leaf of ℓ of T such that for every edge $e \in E(P)$, $\mathbf{w}(e) \leq 2k - 1$. Notice that $\sigma^{-1}(\ell)$ contains exactly one vertex, say x, of S. Moreover, if f is the edge of T that is incident to ℓ, then $\rho(\sigma^{-1}(\ell)) = \mathbf{w}(f) \leq 2k - 1$, as f is an edge of P (the last one). This implies that there is a set of $2k - 1$ edges blocking every path from x to $S \setminus \{x\}$. Therefore, $\mathbf{supp}_G(\infty, x, S \setminus \{x\}) \geq 2k - 1$, contradicting to the fact that S is a $(2k, \infty)$-edge-hideout of G. □

Observation 3.4.15. If H and G are graphs then $H \leq G \Rightarrow \delta_{\mathrm{e}}^{\infty}(H) \leq \delta_{\mathrm{e}}^{\infty}(G)$.

Démonstration. Suppose that $H \leq G$ and that $k \leq \delta_{\mathrm{e}}^{\infty}(H)$. From Theorem 3.3.4 H contains a (k, ∞)-edge-hide-out $S \subseteq V(H)$. Because of Menger's theorem,

Structural Theorems		
	$\not\preceq$ Topological Minor+ Clique-Sum	$\not\preceq$ Immersion+ Edge-Sum
Clique Exclusion	[Grohe et al., 2011] Almost-embedded Almost-bounded-degree	[Wollan, 2015] Almost-bounded-degree
	Vertex	Edge
Bounded Admissibility	[Dvořák, 2012] Almost-bounded-degree	[Limnios et al., 2019] Almost-bounded-degree

TABLE 3.1: Structural Theorems sorted by application and graph property.

for every vertex $v \in S$ there are at least $k+1$ pairwise edge-disjoint paths from v to vertices of $S \setminus \{v\}$. Notice that these paths also exist in G as the "inverse" of the lift operation does not alter the paths from a vertex of S to the rest of the vertices of S. These paths, again using Menger's theorem, imply that S is also a $(k+1, \infty)$-edge-hide-out of G, therefore, again from Theorem 3.3.4, $k \leq \delta_{\mathrm{e}}^{\infty}(G)$. □

We are now ready to give the proof of Theorem 3.4.4.

Proof of Theorem 3.4.4. For the first part of the theorem, observe that $\delta_{\mathrm{e}}^{\infty}(\theta_{k+1}) = k+1$, therefore, from Observation 3.4.15, $\theta_{k+1} \not\preceq G$. Using now the "only if" direction of Theorem 3.4.11 we obtain that $G \in \mathcal{A}_k^{(\leq k)}$, as required.

For the second part of the theorem, let $G \in \mathcal{A}_k^{(\leq k)}$, which by the "if" direction of Theorem 3.4.11 implies that $\theta_{k+1} \not\preceq G$. Using now Lemma 3.4.14, we conclude that $\delta_{\mathrm{e}}^{\infty}(G) \leq 2k-1$. □

3.5 Conclusion

In this chapter, we managed to study as thoroughly as possible edge-degeneracy, and we are able to draw many interesting properties and observations. First it is important to position us toward the existing structural theorems concerning clique exclusion and bounded admissibility that enable us to decompose a graph in almost-bounded-degree graphs. We managed to provide an equivalent result to [Dvořák, 2012] for the edge-admissibility for the graph decomposition using edge-sums Table 3.1.

Finally, in the first part of this chapter, the main result, which motivates us to use edge degeneracy as a new tool for graph decomposition, is the complexity argument. Indeed we proved that, as for the s-degeneracy can be computed in polynomial for $s \in \{1, 2, 3, 4, \infty\}$, the s-edge-degeneracy can be as well computed in polynomial time for $s \in \{1, 2, \infty\}$.

Hence in the next chapter we will study how to design an efficient algorithm for the edge-connectivity degeneracy, and try to compete and compare to k-core mainly and to k-truss for the given application.

CHAPITRE 4

Degeneracy Hierarchy Generators and Efficient Algorithm for Edge Connectivity Degeneracy

In the previous chapter, beyond the structural theorem proven and the complexity guarantees, we set a solid ground to study edge-based degeneracies. It is important to mention that, in Chapter 3, every degeneracy concept is based on the layout of min-max definitions. We showed as well, in Theorem 3.3.4, the equivalent max-min definition for the s-edge-degeneracy, for every path length s, via edge-hideouts. Hence, we have so far equivalent definitions and properties for both s-degeneracy and s-edge-degeneracy.

Therefore, both degeneracy and edge-degeneracy, respectively denoted by δ and δ_e, are defined by means of layouts and are characterized as degree degeneracies. Nevertheless, there exists another degeneracy type that can be defined, derived through the connectivity degeneracy and the edge-connectivity degeneracy, respectively denoted by λ and λ_e. Indeed, instead of defining the degeneracy using layouts and hideouts, we can have the same properties when defining connectivity-degeneracy using carving decompositions and block numbers.

Thus, we will first provide all the needed definitions for theses hierarchy

types, and then study the relationships between degeneracy and connectivity degeneracy. Finally, we propose an efficient algorithm to compute the λ_e^s-edge-connectivity degeneracy, for the particular value of $s = 1$. This could be done mainly thanks to the relationships issued between δ_e^s and λ_e^s and clever use of the Karger-Stein algorithm [Karger and Stein, 1996] for minimal cuts. Lastly, in order to evaluate the quality of one of the connectivity degeneracies issued, we will study interesting properties of the produced decomposition and comment on the observed results on applications where traditional degeneracies shines.

4.1 Introduction

Edge-connectivity degeneracy was already the focus of data mining applications in the litterature. Indeed, both [Zhou et al., 2012] and [Dory, 2018] capitalized on the very interesting structural properties of the k-edge-connected subgraphs, providing more expressive subgraphs than standard vertex degeneracy. More generally, in graph theory, connectivity is a fundamental subject and has applications in a variety of traditional areas. Indeed, a wide range of its application is observed in research fields, such as network reliability analysis, VLSI chip design, transportation planning.
We remind that a k-edge-connected graph is a connected graph that cannot be disconnected by removing less than k edges. Similarly, a k-vertex-connected graph is a connected graph that cannot be disconnected by removing less than k vertices.

In this chapter we will start by redefining both the results and the hierarchy, used and proved in [Richerby and Thilikos, 2011] for vertex degeneracy, and complete these with the results from Chapter 3. Hence, we manage to provide a complete analysis and ordering of the hierarchies based on layouts and hideouts min-max, as well as equivalent max-min definitions. For every degeneracy type, be edge or vertex one, we have an associated admissibility that we will define properly for any path length s. Note that in the previous chapter, we were interested only in the ∞-degeneracy type. Also, no proposed

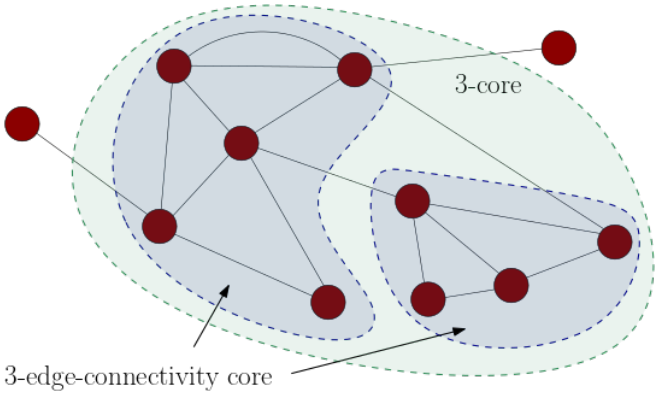


FIGURE 4.1: $\lambda_e^1(G) = 3$ in blue, $\delta^1(G)$ in green

definition was independent of the degeneracy one, since ∞-degeneracy and ∞-admissibility are equal thanks to Menger's min-max theorem. This is also true for the edge version of both the degeneracy and the admissibility.
More importantly we provide a generator for the following eight different hierarchies of invariants, that we shall name :

1. s-degeneracy and associated s-admissibility
2. s-edge degeneracy and associated s-edge admissibility
3. s-connectivity degeneracy and associated s-connectivity admissibility
4. s-edge-connectivity degeneracy and associated s-edge-connectivity admissibility

The first point was thoroughly investigated in [Richerby and Thilikos, 2011], also, we already studied extensively the second one in the first chapter of our dissertation, and the third one will only be defined, not studied as it was investigated in [Kirousis and Thilikos, 1996]. Our work, especially in the practical second part of this chapter, will focus on the edge-connectivity degeneracy.

In the first part of this chapter though, we will carefully define these hierarchies, and provide as much background information and knowledge as possible, that can link them and generalize their design.

4.2 Preliminary Definitions

All graphs in this chapter are undirected, unweighted, loopless and may contain parallel edges as the graphs in Chapter 3.

As we explained previously, we will need some further definitions to unroll our work in this chapter, motivated by the need of a general definition for s-admissibility. So we shall retrace the notions defined in the previous chapter and plug the general admissibility definition to it.

4.2.1 s(-edge)-degeneracy and s(-edge)-admissibility

We remind that the s(-edge)-degeneracy of a graph G, is roughly defined as the largest minimal (e)sep$_{G,s}(x, S)$, where (e)sep$_{G,s}(x, S)$ is the s(-edge)-support of a vertex x in G defined in Subsection 3.3.2, over all layouts in G (see Subsection 3.3.2 for a more detailed definition). Additionally, define the (e)path$_{G,s}(x, S)$ to be the maximum number of (edge-)paths in G starting from x and sinking in S for a given layout L.

So far all (edge-)degeneracies were defined using (e)sep. So, replacing (e)sep by (e)path in the definition of the s(-edge)-degeneracy, we obtain the s(-edge)-admissibility as follows :

s(-edge)-admissibility. Let G be a graph and let $L = \langle v_1, \ldots, v_r \rangle$ be a layout (i.e. linear ordering) of its vertices. Given an $i \in [r]$, we denote $L_{\leq i} = \langle v_1, \ldots, v_i \rangle$. Given an $s \in \mathbb{N}^+_{\geq 1}$, the *$s$(-edge)-admissibility* of L, is the maximum (e)path$_{G,s}(v, L_{\leq i})$ of a vertex v in L. The *s(-edge)-admissibility* of G, denoted by $\alpha^s_{(e)}(G)$ is the minimum s(-edge)-admissibility over all layouts of G.

Hence, we can build four different hierarchies with the same procedure by picking one of the following graph metrics from the following set :

$$\mathcal{M} = \{\text{sep}_{G,s}, \text{esep}_{G,s}, \text{path}_{G,s}, \text{epath}_{G,s}\} \tag{4.1}$$

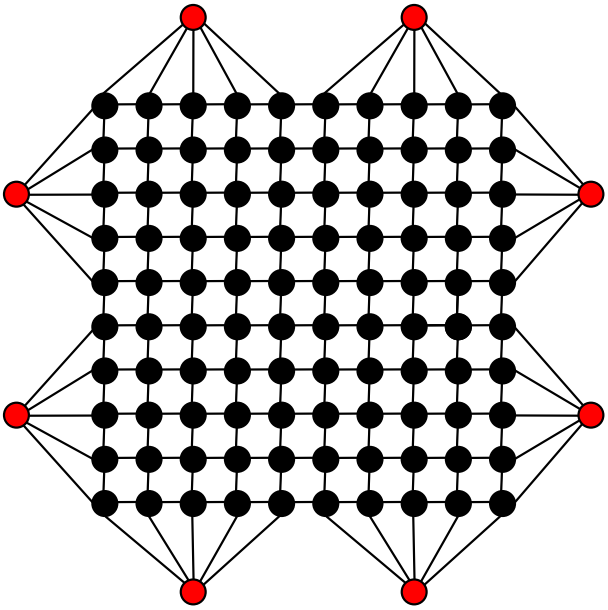


FIGURE 4.2: The 6-outer vertices form a 5-block

4.2.2 k-blocks and Carving Decompositions

We shall define some concepts we are not familiar with yet, starting with the k(-edge)-block.

Definition 4.2.1 (k(-edge)-block)**.** A k(-edge)-block in a graph G is a maximal set of at least k vertices no two of which can be separated in G by deleting fewer than k vertices for the k-block, k edges for the k-edge-block analogue. The (edge-)block number G is the maximum integer k for which G contains a k(-edge)-block.

We have as well to define a carving decomposition of a graph, i.e. we defined in Subsection 3.4.3, Definition 3.4.13, the rooted carving decomposition to be a pair $(\mathbf{T}, \sigma)$ consisting of a rooted binary tree $\mathbf{T} = (T, r)$ and a function $\sigma : V(G) \to L(T)$, we also remind that given $L(T)$ is the set of *leaves* of T. More generally a carving decomposition is a pair $(\mathbf{T}, \sigma)$ where $\mathbf{T}$ is a tree not necessarily rooted or binary and σ a separation function.

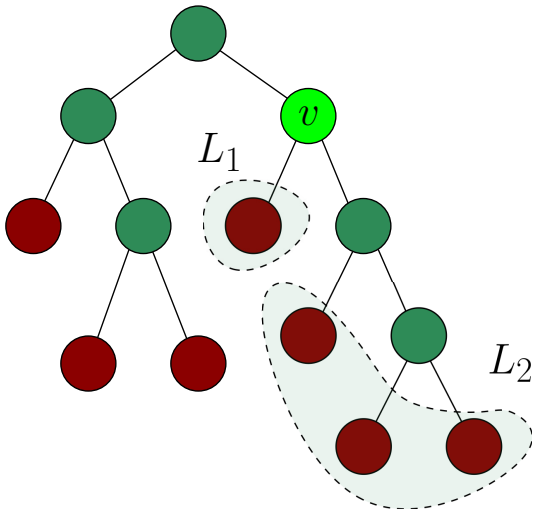


FIGURE 4.3: Example of carving decomposition rooted binary tree, in vertices in green belong to $I(T)$ whereas the red ones are the leaves.

4.2.3 s(-edge)-connectivity Degeneracy and s(-edge)-connectivity Admissibility

In order to build the connectivity degeneracy framework, we need metrics equivalent to $\mathcal{M}$. Indeed, now that we are not working with layouts anymore but with carving decompositions, i.e. trees, we need the metrics in $\mathcal{M}$ to be adapted from single vertex sourced to "from bags to bags". This means that our new set $\mathcal{M}^{(c)}$ contains the following metrics :

$$\mathcal{M}^{(c)} = \{\mathsf{sep}^{(c)}_{G,s}, \mathsf{esep}^{(c)}_{G,s}, \mathsf{path}^{(c)}_{G,s}, \mathsf{epath}^{(c)}_{G,s}\}, \tag{4.2}$$

where (e)$\mathsf{sep}^{(c)}_{G,s}(X_1, X_2)$ is the minimum number of edges/vertices needed to remove in order to separate the set of nodes X_1 from X_2. Similarly, (e)$\mathsf{path}^{(c)}_{G,s}(X_1, X_2)$ is the maximum number of (edges-)paths that connects the set of nodes X_1 to the set X_2.

Min-max Definition for Connectivity Degeneracy and admissibility Let $(\mathbf{T}, \sigma)$ be a carving decomposition and consider the set $\mathcal{M}^{(c)}$ as previously defined. For now on, we make the assumption that every tree is binary and rooted. Define also $(T) = V(T) \setminus L(T)$. Moreover, if $v \in I(T)$, consider both $L_1(v)$ and $L_2(v)$,

the sets leaves issued by v (see Figure 4.3). Then, for a given carving decomposition $(\mathbf{T}, \sigma)$, we have the two following functions :

$$\forall \mu \in \mathcal{M}^{(c)},\ \forall v \in I(T) : \text{cost}_\mu(v) = \mu\Big(\sigma^{-1}\big(L_1(v)\big), \sigma^{-1}\big(L_2(v)\big)\Big) \tag{4.3}$$

and,

$$\text{width}_\mu(T, \sigma) = \max\{\text{cost}_\mu(v) | v \in I(T)\}. \tag{4.4}$$

Finally we can define the s(-edge)-connectivity degeneracy, denoted by $\lambda^s_{(e)}$, as the minimum $\text{width}_\mu(T, \sigma)$ over all rooted carving decompositions with $\mu = \text{(e)sep}^{(c)}_{G,s}$; and equivalently, the s(-edge)-connectivity admissibility, denoted $\gamma^s_{(c)}$ as the minimum width_μ with $\mu = \text{(e)path}^{(c)}_{G,s}$.

Max-min Equivalence for the Connectivity Degeneracy. As for the degree degeneracy, there exists a max-min version of the previous definition. Indeed as we showed in Subsection 3.3.2, there are two ways to define degree degeneracy. The first one corresponds to a min-max definition with layouts, whereas the other one is a max-min characterization by means of hideouts.
In the connectivity framework, this equivalence exists as well, where k-blocks, defined in Definition 4.2.1, are used instead of hideouts. Indeed, consider $S \subseteq V(G)$, so that :

$$\text{cost}^*_\mu(S) = \min\{\mu(X_1, X_2) | X_1, X_2 \subseteq S : X_1 \cup X_2 = S, X_1 \cap X_2 = \emptyset\}. \tag{4.5}$$

If $\text{cost}^*_\mu(S) \leq k$ for a given $\mu = \text{(e)sep}^{(c)}_{G,s}$, therefore, S is a k(-edge)-block. Then, using the proof of Lemma 3.4.14, s(-edge)-connectivity degeneracy is defined as the maximum $\text{cost}^*_\mu(S)$ for every subset S of $V(G)$, given that $|S| > 1$.

Now that the various degeneracy hierarchies are defined in a unified way, the subsequent analysis will focus on deriving properties to link the latter notions.

Also, we show that most of the properties for a given hierarchy can be inherited by another one.

4.3 Connectivity Degeneracy versus Degree Degeneracy

This section derives a unified framework for generating degeneracy hierarchies, based on both degree or connectivity measures. The proposed design is applied on both graphs and multigraphs, *i.e.* graph containing parallel edges. Finally, a classification, as well as an ordering of the hierarchies, will be developed. The goal of this work is to provide as many generic properties as possible, to render feasible the usage of whichever degeneracy concept we might find interest in. Precisely, the latter choice can be mainly driven by the applications at stake. For instance, we will propose in the following section, an in-depth application to edge-connectivity. We will highlight that, when some hierarchies are ordered, these comparisons can lead to very useful heuristics in the design of algorithms to find such degeneracy decompositions.

4.3.1 Degeneracy Hierarchies Generator

We will start by presenting a degeneracy generator, using the sets $\mathcal{M}$ and $\mathcal{M}^{(c)}$. As we saw in the previous section, letting s vary in the s-degeneracy and s-edge-degeneracy, provides us two hierarchy of invariants, i.e. δ^s and δ^s_e. Additionally, consider both the s-admissibility and the s-edge-admissibility as defined in Subsection 4.2.1, respectively defined by α^s and α^s_e.

Hence, similarly to connectivity analysis, both connectivity degeneracies λ^s and λ^s_e, have its corresponding connectivity admissibility γ^s and γ^s_e.
Following the previous section, it is possible to build the latter hierarchies, from $\mathcal{M}$ and $\mathcal{M}^{(c)}$, with the procedure below :

$$\forall \mu \in \mathcal{M},\ \forall s \in \mathbb{N}_{\geq 1} : \quad \delta_\mu = \min_{\forall L} \max_{\substack{v_i \in L \\ i \in \{1,\dots,n\}}} \mu(v_i, L_{\leq i}), \tag{4.6}$$

where L is a layout in G. And for the connectivity analogue, given a carving decomposition :

$$\forall \mu \in \mathcal{M}^{(c)},\ \forall s \in \mathbb{N}_{\geq 1} : \quad \lambda_\mu = \min_{\forall (\mathbf{T},\sigma)} \max_{v \in I(\mathbf{T})} \mu\Big(\sigma^{-1}\big(L_1(v)\big), \sigma^{-1}\big(L_2(v)\big)\Big) \tag{4.7}$$

where $(\mathbf{T}, \sigma)$ chosen among all the rooted carving decompositions of G.

Now, thanks to Equation 6.1 and Equation 4.7, we are able to generate any of the previously mentioned hierarchies with this min-max routine.
It remains to analysis the common properties and possible relationships. In fact we will first present the ordering of the degree vertex degeneracy and admissibility, and try to extend it to the connectivity hierarchies.

4.3.2 s-degeneracy and s-admissibility Comparison

In [Richerby and Thilikos, 2011], the authors worked on the first degree degeneracy hierarchies presented, i.e. vertex s-degeneracy and admissibility. Let us remind the following results and inequalities. By construction we have first :

$$\delta^1 \leq \delta^2 \leq \cdots \leq \delta^\infty \tag{4.8}$$

$$\alpha^1 \leq \alpha^2 \leq \cdots \leq \alpha^\infty. \tag{4.9}$$

Then, for s from 1 to 3 and for $s = \infty$ (proved in [Richerby and Thilikos, 2011]) we have the following relationship between the two hierarchies :

$$\begin{array}{ccccccccc} \delta^1 & \leq & \delta^2 & \leq & \delta^3 & \leq & \ldots & \leq & \delta^\infty \\ \| & & \| & & \| & & \vee | & & \| \end{array} \tag{4.10}$$

$$\alpha^1 \leq \quad \alpha^2 \leq \alpha^3 \leq \cdots \leq \alpha^\infty. \tag{4.11}$$

In Chapter 3, this relationship was proved to hold as well for the edge counterpart of the degree degeneracy, for $s = \infty$. Hence, as this result is known for s from 1 to 2 [Exoo, 1983], similarly :

$$\begin{array}{ccccccc} \delta_e^1 & \leq & \delta_e^2 & \leq \ldots \leq & \delta_e^\infty \\ \| & & \| & \vee | & \| \\ \alpha_e^1 & \leq & \alpha_e^2 & \leq \cdots \leq & \alpha_e^\infty. \end{array} \tag{4.12}$$

$$\tag{4.13}$$

It is clear that it is sufficient to only use δ_e^∞ instead of defining the general concept of admissibility in the latter section.

Finally, such properties are of interest as well for the connectivity hierarchies. Unfortunately, most of these results are unknown so far, but we are confident to conclude with the same tools and techniques used to prove the relationship between Equation 4.12 and Equation 4.13. So far we can ensure the following result :

$$\begin{array}{ccccccc} \lambda_e^1 & \leq & \lambda_e^2 & \leq \lambda_e^3 & \leq \cdots \leq & \lambda_e^\infty \\ \| & & & & & \| \\ \gamma_e^1 & \leq & \gamma_e^2 & \leq \gamma_e^3 & \leq \cdots \leq & \gamma_e^\infty. \end{array} \tag{4.14}$$

$$\tag{4.15}$$

Remark : We are convinced that this type of relationship is true for every hierarchy defined so far but the proofs are left for future work resulting in the following conjecture.

Conjecture 4.3.1. Considering the hierarchy of invariants λ^s/α^s and λ_e^s/α_e^s, namely the s-connectivity degeneracy/admissibility and the s-edge-connectivity degeneracy/admissibility. The following orderings hold :

$$\begin{array}{ccccccc} \lambda^1 & \leq & \lambda^2 & \leq \lambda^3 & \leq \ldots \leq & \lambda^\infty \\ \| & & \| & \| & \vee | & \| \\ \gamma^1 & \leq & \gamma^2 & \leq \gamma^3 & \leq \cdots \leq & \gamma^\infty. \end{array} \tag{4.16}$$

$$\tag{4.17}$$

$$\begin{array}{ccccccc} \lambda_e^1 & \leq & \lambda_e^2 & \leq \ldots \leq & \lambda_e^\infty \\ \| & & \| & \vee | & \| \\ \gamma_e^1 & \leq & \gamma_e^2 & \leq \cdots \leq & \gamma_e^\infty. \end{array} \tag{4.18}$$

$$\tag{4.19}$$

4.3.3 Relation Between Degree and Connectivity

Now that we have a comparison of every degeneracy hierarchy compared with its analogue admissibility, this section attempts to put in relation connectivity and degree degeneracy.

First a straightforward result coming from Lemma 3.4.14 is the following inequality.

Corollary 4.3.2. *For every undirected, loopless, unweighted, with potential parallel edges graph G we have :*

$$\lambda_e^{\infty}(G) \leq \delta_e^{\infty}(G) \leq 2\lambda_e^{\infty}(G) \tag{4.20}$$

Moreover we can prove easily that this result is true for $s = 1$, i.e. we have the following theorem :

Theorem 4.3.3. *For every undirected, loopless, unweighted, with potential parallel edges graph G we have :*

$$\lambda_e^{1}(G) \leq \delta_e^{1}(G) \leq 2\lambda_e^{1}(G) \tag{4.21}$$

Démonstration. Proving that $\delta_e^1(G) \leq 2\lambda_e^1(G)$ is as well a direct result from the proof of Lemma 3.4.14. For the first inequality in Equation 4.21 suppose there exists a layout L_{opt} of G such as $\delta_e^1(G) = \max_{v_i \in L_{opt}} \text{esep}_{1,L_{opt}}(v_i, L_{opt_{\leq i}}) = k$. Suppose now the argmax of this formula is v_m . There exists a tree T_m with root v_m such that width$(T_m, \sigma) = \text{esep}^{(c)}_{(1,L_{opt})}(\{v_m\}, L_{opt} \setminus v_m) = k$ by construction of v_m. Hence, as $\lambda_e^1(G)$ is the minimum value of the width over all possible carving decompositions, $\lambda_e^1(G) \leq k$, thus $\lambda_e^1(G) \leq \delta_e^1(G)$. □

Finally we managed to show that we can rank degeneracies. Here we can notice as well that the $\delta_e^1(G)$ decomposition corresponds to the well known k-core decomposition [Seidman, 1983b]. Hence we can use this theorem the other way around to bound the λ_e^1 degeneracy in real world scenarios using the

k-core algorithm, i.e., $\delta_e^1(G)/2 \leq \lambda_e^1(G) \leq \delta_e^1(G)$. Thus, in the following section, we provide an algorithm to find λ_e^1 using what we learnt so far, and evaluate this degeneracy compared to the known ones.

4.4 Algorithm Design for k-Edge-Connectivity Degeneracy

For now on we will refer to the $\delta_e^1 = \delta^*$ degeneracy decomposition as the k-core decomposition, and similarly denote $\lambda_e^1 = \lambda^*$ degeneracy as the k-edge-connectivity degeneracy decomposition, or k-edge-connectivity cores.

Edge-connectivity degeneracy has already drown interest from its own, i.e. papers like [Zhou et al., 2012] or lately [Dory, 2018], provide algorithmic methods to evaluate λ^* and the corresponding subgraph. These papers propose methods with low complexity, the first for standard computing, in $\mathcal{O}(hl|E|)$ where $h, l \ll |V(G)|$, and the second proposes an algorithm in sublinear time using distributed computing. Although having interesting and rather efficient methods to compute the top edge-connectivity core, none of these papers bothered to try applying it to real world applications. They stopped to comparing runtimes with other papers providing exact or approximations of the edge-connectivity core.

Our goal here is to design thanks to Equation 4.21 and another theorem presented later an algorithm that is competitive with the one from [Zhou et al., 2012], since we do not work with distributed methods. And want to investigate if the edge-connectivity degeneracy outperforms k-core frameworks and the like in classic tasks for these decompositions.

4.4.1 Properties and Heuristics

Let's start with an alternative definition for λ^* using minimal cuts from [Bollobás, 2013].

Definition 4.4.1. Let G be a graph. We define the **edge-connectivity degene-**

racy of G as a follows :

$$\lambda^*(G) = \max\{\lambda(H) \mid H \subseteq G\}.$$

where $\lambda(H) = \min\{cut(X_1, X_2) | X_1 \cup X_2 = V(H), X_1 \cap X_2 = \emptyset\}$ giving the minimal cut in H.

Following this definition we will not need carving decompositions or k-blocks to compute λ^*. Minimal cuts, as a k-edge-connected graph can not be split by a minimal cut of size less than k. A naive way would then be to check the minimal cut between every pair of nodes to find λ^*.
Unfortunately, computing minimal cuts in a graph can be done in $\mathcal{O}(n^2 \log(n))$ with the **Karger-Stein** algorithm [Karger and Stein, 1996], which means that finding the edge-connectivity core can be done in $\mathcal{O}(n^2(m + n\log(n)))$ with an adapted algorithm from [Yan et al., 2005], which is terrible. Especially if we compare to the algorithm from [Zhou et al., 2012], where the complexity of its algorithm is $\mathcal{O}(n^2 \log(n))$.

The major problem of this method is the need to compute minimal cut algorithm in the whole graph. Thankfully thanks to Equation 4.21 we know that we do not need to look at the whole graph to find λ. We also have the following theorem from [Menger, 1927] up our sleeve that we will benefit from :

To tackle the problem of computing naively the mincut algorithm between every pair of nodes in a graph. We need to find some heuristic to explore only a small part of the graph. Indeed such a heuristic can be found thanks to Equation 4.21 and the following theorem.

Theorem 4.4.2. *Let G be a graph and let $\mathcal{C} = \{C_1, \ldots, C_r\}$ be a collection of vertex disjoint connected subgraphs of G. Let also G' be the graph obtained if we contract in G all edges in the graphs in $\mathcal{C}$. If G' is d-edge connected and each graph in $\mathcal{C}$ is d-edge connected or a single vertex, then G contains a subgraph that is d-edge connected.*

Consider now a graph G, following Equation 4.21, the top k-edge-connectivity

core is somewhere between $\delta^*(G)/2$ and $\delta^*(G)$. Therefore, a first search criterion would be to look for the top k-edge-connectivity core in subgraphs between the $\delta^*/2$-core and δ^*-core of a graph. This key step leads to computing minimal cuts in subgraphs that are consistently smaller than the original graph. This trick enables us to reduce drastically the complexity of the method using minimal cuts to $\mathcal{O}(n_{k/2}^3 \log^3(n_{k/2}))$ in the worst cases, and in the best cases $n_k \ll n$ making the method very fast.

4.4.2 Algorithm Design

Algorithm 4 Contraction Algorithm

```
1: procedure CONTRACT(G, nodes)
2:     Input G : Undirected Graph, nodes : Nodes from a subgraph to contract
3:     Output G : Transformed Graph with contracted subgraph
4:
5:     sinknode = nodes[0]
6:     for node ∈ nodes[1 :] do
7:         G.neighbors[sinknode] += G.neighbors[node]
8:         G.remove(node)
9:     end for
10: end procedure
```

Although the previous reduction is a very efficient way to reduce the computational cost, the worst case scenario being having to look up to the $\delta^*/2$-core remains expensive. This subgraph can contain a third of the nodes in some cases which is not an interesting improvement anymore. Hence thanks to a recursive application of the contraction algorithm 4 and the previous trick, we are able to propose an algorithm that runs, in the worst case scenario, in $\mathcal{O}(\frac{k}{2} n_{k/2}^2 \log^2(n_{k/2}))$ for finding the top k-edge-connectivity core.

k-edge-connectivity core finding procedure. Let $k = \delta^*(G)$. First, $k/2 \leq \lambda^*(G) \leq k$. For every $z \in [k/2, k]$ (starting from k), the z-edge-connectivity core can be found using the procedure :

— **Initialisation :** Set `found` $=$ FALSE.

— **Step 1 :** Let A_k be the k-core of G.

— **Step 2 :** Run the Karger-Stein algorithm on A_k in order to find an edge cut of size $< z$. If no such set is found, then it is **known** that $\lambda^*(A_k) \geq z \Rightarrow \lambda^*(G) \geq z$. If such a cut is found for some $S \subseteq V(A_k)$, set $\texttt{found} = \text{TRUE}$, $A_k^{(1)} = A_k[S]$ and $A_k^{(2)} = A_k \setminus S$. Recursively run the Karger-Stein algorithm on $A_k^{(1)}$ and $A_k^{(2)}$. When this recursion finishes one obtains a partition of A_k into sets $\mathcal{S} = \{S_1, \ldots, S_q\}$, where each S_i either is a singleton or induces a z-edge-connected subgraph in A_k.

— **Step 3 :** If $\mathcal{S}$ is not trivial, it is known that $\lambda^*(A_k) \geq z \Rightarrow \lambda^*(G) \geq z$. Then, construct G' by contracting all non-trivial S_i's. Denote by G' the resulting graph. Notice that, by Theorem 4.4.2, $\lambda^*(G') \geq z \Leftrightarrow \lambda^*(G) \geq z$. Then set $G := G'$, $\texttt{found} := \text{TRUE}$ and go to step 1.

— **Step 4 :** If $\mathcal{S}$ is trivial and $\texttt{found} = \text{TRUE}$, use the essential singletons in $\mathcal{S}$ to build the z-edge core partition and stop.

— **Step 5 :** If $\mathcal{S}$ is trivial and $\texttt{found} = \text{FALSE}$, set $k := k - 1$ and go to step 1.

Remark : This algorithm is guaranteed to terminate thanks to Equation 4.21.

It is important to note that this algorithm can be used not only to find the k-edge-connectivity core, but also the remaining decomposition of the edge-connectivity core. Indeed, it can be achieved by contracting the previous core found in the initial graph and searching for the top k-edge-connectivity core in this new graph. Of course, the more k will be close to 0, the higher the complexity.

4.5 Experiments

In this section we will investigate possible use of the edge-connectivity cores and compare the obtained carving decomposition tree to the k-core nested decomposition. We will at first show the inherent difference between the k-edge-connectivity core and the k-core decomposition. In the second part, we will discuss the results on seeding spreading processes, to compare to similar frameworks such as k-truss decomposition.

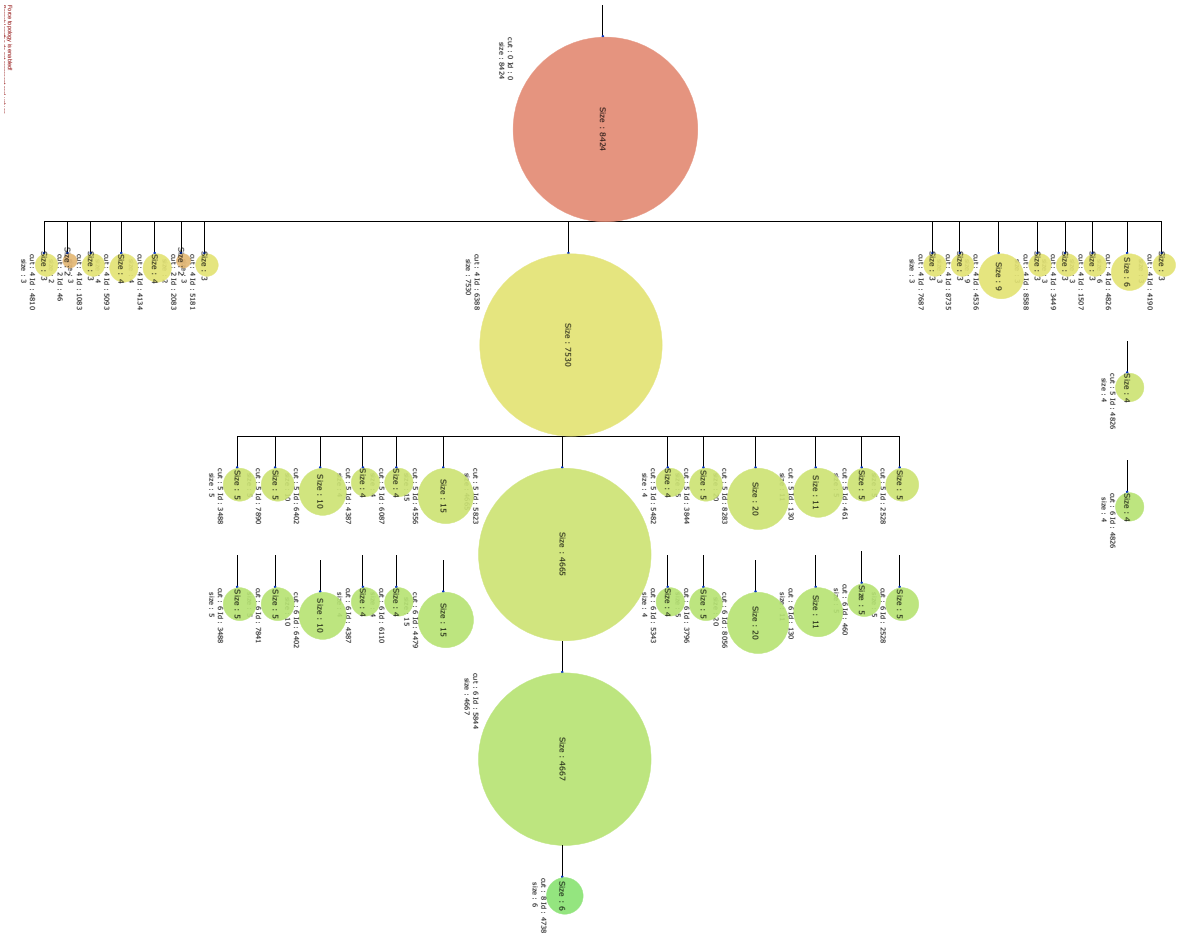


FIGURE 4.4: Khaufold graph edge-connectivity carving tree decomposition. Every circle represents a subgraph, and is logarithmically proportional to its size. A connection means that it is contained in the subgraph above. A particular color represents a given k-edge-connectivity core, where $\lambda^*(G) = 8$ in darker green up to 0 in red corresponding to the whole graph.

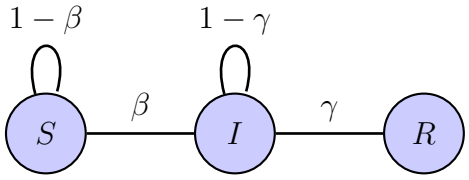
4.5.1 Tree-like Decomposition versus Nested One

It is known that the k-core decomposition, as shown in Figure 2.3, provides a linear nested layout decomposition of the graph. Briefly, for a given connected component, the k-core is a subgraph of the $k-1$-core, and so on. Also, every core is a connected component nested in the previous one. This is not necessarily the case for the k-edge-connectivity cores, as shown in Figure 4.4. Unfortunately this figure is quite rare *i.e.* in the real world datasets, especially the ones without parallel edges do not present this kind of tree-like carving. Most of the real datasets have **linear layouts**, that is almost the same as the

one provided by the k-core decomposition.

4.5.2 Epidemic Models

As k-truss decomposition shined in finding influential spreaders in epidemic models, we first wanted to compare our framework to k-core and k-truss in this domain. It was also emphasised by Figure 4.4, that some nodes situated in the joints of the trees, *i.e.* that have lots of close connections to other leafs, may be very efficient spreaders. It is fairly obvious that seeding an infection in any k-edge-connectivity core ensures a very fast spreading of this infection within the given leaf it has been implanted in. A natural selection for spreads will then be nodes that are on bridges between these leafs and also contained in the most connected ones. Hence we will evaluate the spreaders with one of the mostly studied epidemic models, the **Susceptible-Infected-Recovered** (SIR) model [Barrat et al., 2008]. The compartmental model assumes a population of N individuals, divided into three states :



1. *Susceptible (S)* : The individual is not yet infected, thus being susceptible to the epidemic.
2. *Infected (I)* : The individual has been infected with the disease and it is capable of spreading the disease to the susceptible population.
3. *Recovered (R)* : After an individual has experienced the infectious period, it is considered as removed from the disease and it is not able to be infected again or to transmit the disease to others (immune to further infection or death).

Moreover, an *infected* node can infect a neighbor with probability β at each time step. The *infected* node can also recover with probability γ. For our experiments those two constants are fixed for each given dataset. We choose β to

be close to the epidemic threshold given by $\tau = 1/\lambda_1$ where λ_1 is the largest eigenvalue of the given graph adjacency matrix, γ is arbitrarily set to 0.8.

4.5.3 Datasets

We evaluate our spreaders on two graph datasets, where both are undirected with no parallel edges. The first one, **Email-Enron** is a collaboration network of email exchanges within the Enron Corporation. The second one, **Wiki-vote** contains the voting data from the Wikipedia website. Statistics for both datasets are detailed in Table 4.1. Moreover, for each dataset, we selected the best fit from the influential spreaders in the set $\mathcal{T}$ from the deepest core of each method. For our method, as already mentioned, each node from a given core is connected to the others by at least k edge independent paths. We chose the ones that have the more edges pointing outside the given cores.

Network Name	Nodes	Edges	δ^*	T_{max}	λ^*	$\lvert\mathcal{T}\rvert$	β
EMAIL-ENRON	$33,696$	$180,811$	43	22	43	45	0.01
WIKI-VOTE	$7,066$	$100,736$	53	23	53	50	0.009

TABLE 4.1: Datasets used for the experiments, δ^* is the top k for the k-core decomposition, T_max and λ^* the corresponding ones for the k-truss and the k-edge-connectivity cores. $|\mathcal{T}|$ is the number of influential spreaders initially selected from each method and β is the infection probability.

As we saw, the cores produced by the edge-connectivity have a lot of very interesting properties, such as **the ability to produce very good spreaders**.

Nevertheless, another application of this work can focus on **influence maximisation problems**. For instance, one can find a sufficiently dense/big core and try a greedy algorithm in order **to find the most influential nodes** in those subgraphs (lets say one for each disconnected subgraph). The underlying intuition being that a main influential node, will **influence very quickly** the rest of the subgraph and hence locally maximise the influence.

4.5.4 Results

As explained in the previous sections, a comparison of the selected spreaders from the k-edge-connectivity core to the ones from the k-core and the

	Method	Time Step 2	4	6	8	10	*Final step*	*Max step*
EMAIL-ENRON	**edge core**	21.47	115.13	350.13	428.23	250.93	2,647.74	28
	truss	8.44	46.66	204.08	418.77	355.84	2,596.52	33
	core	4.78	31.97	152.55	367.28	364.13	2,465.60	37
	top degree	6.89	34.13	155.48	360.89	357.08	2,471.67	36
WIKI-VOTE	**edge core**	5.15	12.15	24.72	40.96	52.74	626.09	34
	truss	2.92	6.92	15.27	28.73	42.46	560.66	52
	core	1.92	4.78	10.65	20.66	32.40	466.01	57
	top degree	2.43	5.46	12.05	23.05	35.55	502.88	62

TABLE 4.2: Cumulative number of infected nodes per step of the SIR model using β close to the epidemic threshold of each graph and $\gamma = 0.8$. The Final step column shows the total number of infected nodes at the end of the process (Max step column).

k-truss of each graph, was performed. First, by results in Table 4.2, it is clear that the seeders extracted from the edge-connectivity core infect way faster and spread quickly through the network. Indeed, at every time step of the spreading process we outperform the other degeneracy frameworks.
Unfortunately, even if the results here are very good, we must give some clarifications and drawbacks to our experiments. As it was explained in the previous section, the choice of our spreaders is not just taking random nodes from the top k-edge-connectivity core, there were some characteristics, problem specific, to choose the spreaders among the nodes available. Moreover, the fact that we only have two datasets is due to two main reasons. The first one, as it was developed in Subsection 4.5.1, both the tree carving decomposition of the k-edge-connectivity core and the layout of the k-core were **almost the same**. This is mainly due to the fact that meaningful graph datasets with sets of parallel edges, where the k-edge-connectivity degeneracy really shines are hard to find.

On the other hand, our method is **computationally very expensive**, way more than the k-core and k-truss method. Not only that, as we said, the decomposition for most datasets the k-edge-connectivity decomposition is very close, if not the same, as the one issued from the k-core. Even in cases like 4.5 where it looks like there is an important difference between the two decompositions, looking closer we see that it does not provide tremendous information : here

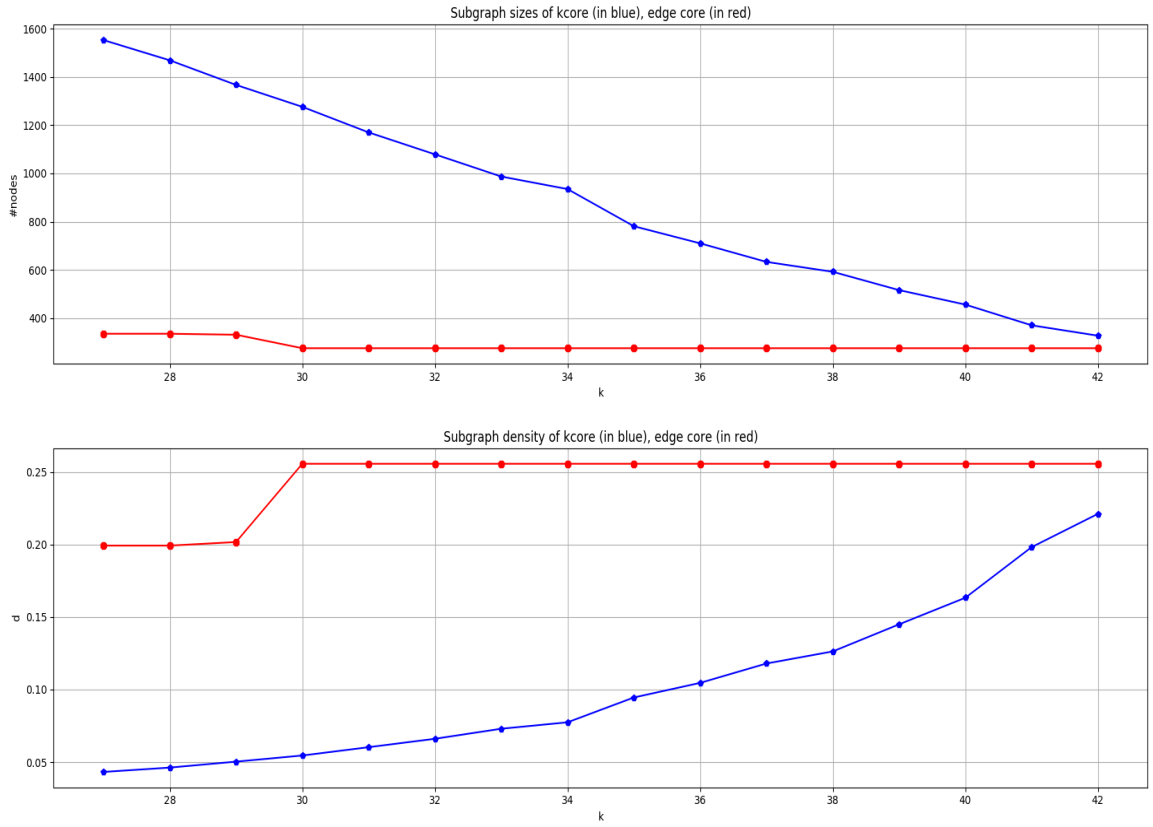


FIGURE 4.5: Size (top graph) and ϵ density (bottom graph) comparison of the edge-connectivity core decomposition (in red) versus the k-core decomposition (in blue), note that it starts at the 42 core instead of 43 which is the top core of the k-core because both cores are the same for $k = 43$.

the top k-edge-connectivity core and the top k-core are the same. And even taking a lower k for the edge-connectivity core will be the same as taking the top k-core, but with a tremendous computational cost in addition to it. Hence we ask ourselves why use the edge-connectivity core and not the k-core.

4.6 Conclusion

In this Chapter, we managed to gather and unify degree degeneracy hierarchies, connectivity degeneracy hierarchies and their corresponding admissibility hierarchies. Hence we are able to generate them with a simple framework of min-max definitions with the four graph metrics we presented. Moreover as we

started giving inequalities and properties ordering degeneracy and corresponding admissibility in Chapter 3, we extended this knowledge and saw that the properties and hierarchies proposed follow all the same behaviour and have relationships that can be generalized. It was also very interesting to see that one can benefit from the other in order to provide bounds for a given connectivity or degree degeneracy. Capitalizing on this knowledge we started to study the edge-connectivity degeneracy as it was the most natural candidate to produce exploitable graph decompositions.

Unfortunately the study of edge-connectivity cores proved to be less impacting than what we expected. Even Though we managed to provide a very efficient algorithm, both real-world dataset structure and density caused our method to perform poorly. The fact, though, that the edge-connectivity core decompositions were very close to the ones provided by the k-core decomposition, gives us more information and motivation to stick with the k-core. Indeed, thanks to [Batagelj and Zaveršnik, 2011] providing a very fast algorithm for k-core decomposition (complexity of $\mathcal{O}(n\log(n))$), and the fact that for most graphs, the k-core decomposition has properties from edge-connectivity degeneracy, *i.e.* not splittable by k size minimal cuts. We can assume that using the k-core decomposition as an approximation to the edge-connectivity degeneracy is an acceptable solution.

Hence, in the following chapters of this dissertation, we will use directly k-core or variants of it for given graph structures to tackle the upcoming problems. As for these problems, k-core was almost exclusively used in an unsupervised learning scope. Usually making for a very good feature, or subgraph extraction. We want to see if this framework can be plugged to supervised learning routines, and one of the first ones we wanted to evaluate it is in graph kernels, as it is a milestone in graph classification.

CHAPITRE 5

A Degeneracy Framework for Graph Similarity

The problem of accurately measuring the similarity between graphs is at the core of many applications in a variety of disciplines. Most existing methods for graph similarity focus either on local or on global properties of graphs. However, even if graphs seem very similar from a local or a global perspective, they may exhibit different structure at different scales. In this chapter, we present a general framework for graph similarity which takes into account structure at multiple different scales.

The proposed framework capitalizes on the k-core decomposition of graphs in order to build a hierarchy of nested subgraphs. We apply the framework to derive variants of four graph kernels, namely graphlet kernel, shortest-path kernel, Weisfeiler-Lehman subtree kernel, and pyramid match graph kernel. Nevertheless, we provide a performance criterion that enable us to predict if the core-kernel variant of a given kernel is able to out-perform the latter.

The framework though is not limited to graph kernels, but can be applied to any graph comparison algorithm. The proposed method is then evaluated on several benchmark datasets for graph classification. In most cases, the core-based kernels achieve significant improvements in terms of classification accu-

racy over the base kernels, while their time complexity remains very attractive.

5.1 Introduction

Graphs are well-studied structures which are utilized to model entities and their relationships. In recent years, graph-based representations have become ubiquitous in many application domains. For instance, social networks, protein and gene regulatory networks, and textual documents are commonly represented as graphs. Furthermore, in the past years, graph classification has arisen as an important topic in many domains such as in Computational Biology [Schölkopf et al., 2004], in Chemistry [Mahé and Vert, 2009] and in Natural Language Processing [Nikolentzos et al., 2017a]. For example, in Chemistry, we are often interested in predicting the mutagenicity of a chemical compound by comparing its graph representation with other compounds of known functionality.

So far, kernel methods have emerged as one of the most effective tools for graph classification, and have achieved state-of-the-art results on many graph datasets [Shervashidze et al., 2011]. Once we define a positive semidefinite kernel function for the input data, a large family of learning algorithms called kernel methods [Smola and Schölkopf, 1998] become available. In more details, kernels are functions that correspond to a dot product in a reproducing kernel Hilbert space, and which measure the similarity between two objects. Kernel functions do not require their inputs to be represented as fixed-length feature vectors, and they can also be defined on structured data such as graphs, trees and strings. Hence, kernel methods provide a flexible framework for performing graph classification.

Most graph kernels in the literature are instances of the R-convolution framework [Haussler, 1999]. These kernels decompose graphs into their substructures and add up the pairwise similarities between these substructures. Specifically, there are kernels that compare graphs based on random walks [Gärtner et al., 2003, Vishwanathan et al., 2010, Sugiyama and Borgwardt, 2015], sub-

trees [Gärtner et al., 2003, Mahé and Vert, 2009], cycles [Horváth et al., 2004], shortest paths [Borgwardt and Kriegel, 2005], and small subgraphs [Shervashidze et al., 2009, Kriege and Mutzel, 2012]. Recently, there was a surge of interest in kernels that are built upon global properties of graphs [Johansson et al., 2014, Johansson and Dubhashi, 2015, Nikolentzos et al., 2017b]. In general, these approaches embed the vertices of each graph in a vector space, and then compare graphs based on these embeddings.

Most existing graph kernels can thus be divided into two classes. The first class consists of kernels that compare local substructures of graphs (i.e. trees, cycles, graphlets), while the second class includes kernels that capture global properties of graphs and are sensitive to the large scale structure of graphs. Some examples of the second class are the random walk based kernels, and the kernels that compare graphs based on the embeddings of their vertices. Therefore, existing graph kernels focus mainly on either local or global properties of graphs. In practice, it would be desirable to have a kernel that can take structure into account at multiple different scales [Kondor and Pan, 2016]. Two well-known kernels that account for that are the Weisfeiler–Lehman subtree kernel [Shervashidze et al., 2011] and the propagation kernel [Neumann et al., 2016]. However, both approaches assume node-labeled graphs. Recently, the multiscale Laplacian kernel was introduced to effectively compare structure at different scales [Kondor and Pan, 2016], while some neural network architectures were also designed to address the same problem [Dai et al., 2016].

In this chapter, we propose a framework for comparing structure in graphs at a range of different scales. Our framework is based on the k-core decomposition which is capable of uncovering topological and hierarchical properties of graphs. Specifically, the k-core decomposition builds a hierarchy of nested subgraphs, each having stronger connectedness properties compared to the previous. By measuring the similarity between the corresponding according to the hierarchy subgraphs and combining the results, we can build more accurate measures of graph similarity.

More specifically, the contributions of this chapter are threefold :

— We propose a general framework that allows existing graph similarity algorithms to compare structure in graphs at multiple different scales. The framework is based on the k-core decomposition of graphs and is applicable to any graph comparison algorithm.

— We demonstrate our framework on four graph kernels, namely the graphlet kernel, the shortest path kernel, the Weisfeiler-Lehman subtree kernel, and the pyramid match kernel.

— We evaluate the proposed framework on several benchmark datasets from bioinformatics, chemoinformatics and social networks. In most cases, the variants obtained from our framework achieve significant improvements over the base kernels.

The rest of this chapter is organized as follows. Section 5.2 introduces some preliminary concepts and gives details about graph degeneracy and the k-core decomposition. Section 5.3 provides a detailed description of our proposed framework for graph similarity. Section 5.4 evaluates the proposed framework on several standard datasets.

5.2 Preliminaries

In this section, we first define our notation, and we then introduce the concepts of k-core and degeneracy. We also give details about the algorithm that extracts the k-cores of a graph.

5.2.1 Definitions and Notations

Let $G = (V, E)$ be an undirected and unweighted graph consisting of a set V of vertices and a set E of edges between them. We will denote by n the number of vertices and by m the number of edges. The neighbourhood $\mathcal{N}(v)$ of vertex v is the set of all vertices adjacent to v. Hence, $\mathcal{N}(v) = \{u : (v, u) \in E\}$ where (v, u) is an edge between vertices v and u of V. We denote the degree

of vertex v by $d(v) = |\mathcal{N}(v)|$.

Given a subset of vertices $S \subseteq V$, let $E(S)$ be the set of edges that have both end-points in S. Then, $G' = (S, E(S))$ is the subgraph induced by S. We use $G' \subseteq G$ to denote that G' is a subgraph of G. The degree of a vertex $v \in S$, $d_{G'}(v)$, is equal to the number of vertices that are adjacent to v in G'.A labeled graph is a graph with labels on vertices and/or edges.

In this chapter, we will consider two types of graphs : (1) unlabeled graphs and (2) graphs with labeled vertices. For the second type of graphs, given a set of labels $\mathcal{L}$, $\ell : V \to \mathcal{L}$ is a function that assigns labels to the vertices of the graph.

5.2.2 Base Kernels

A graph kernel is a symmetric, positive semidefinite function on the set of graphs G. Once we define such a function $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ on the set $\mathcal{G}$, it is known that there exists a map $\phi : \mathcal{G} \to \mathcal{H}$ into a Hilbert space $\mathcal{H}$, such that :

$$k(Gi, Gj) = \langle \phi(G_i), \phi(G_j) \rangle_{\mathcal{H}}$$

for all $G_i, G_j \in G$ where $\langle ., . \rangle_{\mathcal{H}}$ is the inner product in $\mathcal{H}$. Roughly speaking, a graph kernel is a function that measures the similarity of two graphs.

Our framework will be applied on the following graph kernels :

— **graphlet kernel** (GR) [Shervashidze et al., 2009] : The graphlet kernel counts identical pairs of graphlets (i.e. subgraphs with k nodes where $k \in 3, 4, 5$) in two graphs.

— **shortest path kernel** (SP) [Borgwardt and Kriegel, 2005] : The shortest path kernel counts pairs of shortest paths in two graphs having the same source and sink labels and identical length.

— **Weisfeiler-Lehman subtree kernel** (WL) [Shervashidze et al., 2011] : The Weisfeiler-Lehman subtree kernel for a number of iterations counts pairs

of matching subtree patterns in two graphs, while at each iteration updates the labels of the vertices of the two graphs.

— **pyramid match graph kernel** (PM) [Nikolentzos et al., 2017b] : The pyramid match graph kernel first embeds the vertices of the input graphs in a vector space. It then partitions the feature space into regions of increasingly larger size and takes a weighted sum of the matches that occur at each level.

5.3 Degeneracy Framework

In this Section, we propose a new framework for graph similarity that is based on the concept of k-core, and we show how existing graph kernels can be plugged into the framework to produce more powerful kernels.

5.3.1 Core-based Graph Kernels

We next propose a framework for obtaining variants of existing graph kernels. Since the framework utilizes the k-core decomposition, we call the emerging kernels *core variants* of the base kernels. The proposed framework allows the comparison of the structure of graphs at multiple different scales as these are expressed by the graphs' k-cores.

The intuition of using the k-core algorithm to decompose a graph is that internal cores are more important compared to external cores. Hence, they are more likely to reveal information about the class label of the graph compared to external cores. This is by no means implausible since internal cores correspond to subgraphs of high density. As mentioned above, the k-core decomposition is typically used to identify areas of increasing connectedness inside the graph. In almost all graphs, density is an indication of importance [Lee et al., 2010]. For example, in protein-protein interaction networks, dense subgraphs may correspond to protein complexes. Hence, we expect that by decomposing graphs into subgraphs of increasing importance, we will be able to capture their underlying structure, and compare them effectively.

We next introduce the degeneracy framework for deriving core variants of existing kernels.

Definition 5.3.1. Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. Let also k be any kernel for graphs. Then, the core variant of the base kernel k is defined as

$$k_c(G, G') = k(C_0, C'_0) + k(C_1, C'_1) + \ldots + k(C_{\delta^*_{min}}, C'_{\delta^*_{min}}) \tag{5.1}$$

where δ^*_{min} is the minimum of the degeneracies of the two graphs, and $C_0, C_1, \ldots, C_{\delta^*_{min}}$ and $C'_0, C'_1, \ldots, C'_{\delta^*_{min}}$ are the 0-core, 1-core,..., δ^*_{min}-core subgraphs of G and G' respectively.

In the following, we will prove the validity of the core variants produced by our framework.

Theorem 5.3.2. *Let the base kernel k be any positive semidefinite kernel on graphs. Then, the corresponding core variant k_c of the base kernel k is positive semidefinite.*

Démonstration. Let ϕ be the feature mapping corresponding to the base kernel k

$$k(G, G') = \langle \phi(G), \phi(G') \rangle$$

Let $g_i(\cdot)$ be a function that removes from the input graph all vertices with core number less than i and their incident edges. Then, we have

$$k(C_i, C'_i) = \big\langle \phi(g_i(G)), \phi(g_i(G')) \big\rangle$$

Let us define the feature mapping $\psi(\cdot)$ as $\phi(g_i(\cdot))$. Then we have

$$k(C_i, C'_i) = \langle \psi(G), \psi(G') \rangle$$

hence k is a kernel on G and G' and k_c is positive semidefinite as a sum of positive semidefinite kernels.

□

Given two graphs G, G' and a base kernel k, the steps of computing the core variant of k are given in Algorithm 5.

Algorithm 5 Core-based Kernel

procedure CORE-KERNEL(G, G')
  Input : A pair of graphs G and G'
  Output : Result of the kernel function val

  $val = 0$
  $\delta^*_{min} = \min\left(\delta^*(G), \delta^*(G')\right)$
  Let C_i, C'_i be the i-cores of G, G', for $i = 0, \ldots, \delta^*_{min}$
  for $i = \delta^*_{min}$ **to** 0 **do**
    $val = val + kernel(C_i, C'_i)$
  end for
end procedure

The above definition provides a framework for increasing the expressive power of existing graph kernels.

In contrast to other existing frameworks, the proposed framework is not limited to R-convolution kernels [Yanardag and Vishwanathan, 2015] or to node-labeled graphs [Shervashidze et al., 2011]. Furthermore, it should be mentioned that the proposed framework is not even restricted to graph kernels, but can be applied to any algorithm that compares graphs. Hence, it can serve as a generic tool applicable to the vast literature of graph matching algorithms [Conte et al., 2004].

5.3.2 Computational Complexity

The proposed framework takes into account structure at different scales, yet it remains an interesting question how it compares to base kernels in terms of runtime complexity.

Its computational complexity depends on the complexity of the base kernel and the degeneracy of the graphs under comparison. More specifically, given a pair of graphs G, G' and an algorithm A for comparing the two graphs, let $\mathcal{O}_A$ be the time complexity of algorithm A.

Let also $\delta^*_{min} = \min\left(\delta^*(G), \delta^*(G')\right)$ be the minimum of the degeneracies of the two graphs. Then, the complexity of computing the core variant of algorithm A is $\mathcal{O}_c = \delta^*_{min}\mathcal{O}_A$. It is well-known that the degeneracy of a graph is upper

bounded by the maximum of the degrees of its vertices and by the largest eigenvalue of its adjacency matrix λ_1. Since in most real-world graphs it holds that $\lambda_1 \ll n$, it also holds that $\delta^*_{max} \ll n$, and hence, the time complexity added by the proposed framework is relatively low.

Moreover some base kernels, one might be able to exploit the fact that high-order cores are contained into lower-order cores in order to perform some computations only once instead of repeating them for all cores. One example of such a base kernel is the graphlet kernel. Given two cores of a graph C_i and C_j with $i < j$, all the graphlets found in C_j will also be present in C_i.

5.3.3 Dimensionality Reduction Perspective

The k-core decomposition can also be seen as a method for performing dimensionality reduction on graphs. Given the i-cores C_i, $i = 1, \ldots, \delta^*(G)$ of a graph G, each core C_i can be considered as an approximation of the graph where features of low importance (i.e. vertices belonging to low-order cores and their incident edges) have been removed from the graph. The approximation error can be computed by the Frobenius norm of the difference of the adjacency matrices of the two graphs $er = ||A - A_i||_F$ where A, A_i are the adjacency matrices of graph G and its i-core respectively.

In cases where the input graphs are very large, the running time of high-complexity algorithms is prohibitive. For example, computing the shortest path kernel on the D&D dataset takes almost 1 hour. In such cases, we can take advantage of the k-core decomposition to effectively prune a large number of vertices from the input graphs by retaining only their high-order cores. Then, it may be possible to employ a high-complexity algorithm. For example, by replacing the graphs contained in the D&D dataset with their 3-cores, we managed to compute the core variant of the shortest path kernel in less than 5 minutes and to achieve accuracy comparable to the best performing algorithms ($avg.\ acc = 77.92$).

5.4 Experiments and Evaluation

In this section, we first describe the datasets that we used for our experiments. We next give details about the experimental settings. We last report on the performance of the base kernels and the core variants.

5.4.1 Datasets

We evaluated the proposed framework on standard graph classification datasets derived from bioinformatics and chemoinformatics (MUTAG, ENZYMES, NCI1, PTC-MR, D&D), and from social networks (IMDB-BINARY, IMDB-MULTI, REDDIT-BINARY, REDDIT-MULTI-5K, REDDIT-MULTI-12K) [1]. Note that the social network graphs are unlabeled, while all other graph datasets come with vertex labels.

5.4.2 Experimental Setup

To perform graph classification, we employed a C-Support Vector Machine (SVM) classifier and performed 10-fold cross-validation. The whole process was repeated 10 times for each dataset and each method. The parameter C of the SVM was optimized on the training set only.

All kernels were written in Python [2]. The parameters of the base kernels and their corresponding core variants were selected using cross-validation on the training dataset. We chose parameters for the graph kernels as follows. For the graphlet kernel, on labeled graphs, we count all connected graphlets of size 3 taking labels into account, while on unlabeled graphs, we sample 500 graphlets of size up to 6. For the Weisfeiler-Lehman subtree kernel, we chose the number of iterations h from $\{4, 5, 6, 7\}$. For the pyramid match kernel, the dimensionality of the embeddings d was chosen from $\{4, 6, 8, 10\}$, while the number of levels L was chosen from $\{2, 4, 6\}$.

1. The datasets and statistics are available at https ://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets
2. Code available at https ://www.lix.polytechnique.fr/ nikolentzos/code/core_framework.zip

We report in Table 5.1 average prediction accuracies and standard deviations. Core variants with statistically significant improvements over the base kernels are shown in bold as measured by a t-test with a p value of ≤ 0.05. We also report in Table 5.2 the time required for computing the kernel matrix of each core variant relative to the time required for computing the kernel matrix of its base kernel as measured on a 3.4GHz Intel Core i7 with 16Gb of RAM.

5.4.3 Results

We begin our experiments by comparing the base kernels with their core variants. Table 5.1 demonstrates that the proposed framework improves the classification accuracy of every base kernel on almost all datasets.

More specifically, the core variants outperformed their base kernels on 37 out of the 40 experiments. It should be mentioned that the difference in performance between the core variants and their base kernels was larger on the social interaction datasets compared to the bioinformatics and chemoinformatics datasets.

The obtained results confirm our intuition that the densest areas of graphs are the most important. Furthermore, the results show that the hierarchy of nested subgraphs generated by the k-core decomposition allows existing algorithms to compare structure in graphs at multiple different scales. On most datasets, the increase in performance of the GR, SP and PM kernels due to the use of the proposed framework is very large.

Specifically, core GR improved by more than 10% the accuracy attained by the GR kernel on 4 datasets. Conversely, core WL yielded in general only slightly better accuracies compared to its base kernel. The WL kernel builds a summary of the neighborhood of each vertex.

Our intuition is that the local neighborhood of a vertex in a k-core is not dramatically different from its neighbourhood in the graph. Hence, for small values of the parameter h of WL, the summaries that are generated in a k-core are very similar to those generated in the whole graph and do not thus provide much additional information.

Method \ Dataset	MUTAG	ENZYMES	NCI1	PTC-MR	D&D
GR	69.97 (± 2.22)	33.08 (± 0.93)	65.47 (± 0.14)	56.63 (± 1.61)	77.77 (± 0.47)
Core GR	**82.34** (± 1.29)	33.66 (± 0.65)	**66.85** (± 0.20)	57.68 (± 1.26)	78.05 (± 0.56)
SP	84.03 (± 1.49)	40.75 (± 0.81)	72.85 (± 0.24)	60.14 (± 1.80)	77.14 (± 0.77)
Core SP	**88.29** (± 1.55)	41.20 (± 1.21)	**73.46** (± 0.32)	59.06 (± 0.93)	77.30 (± 0.80)
WL	83.63 (± 1.57)	51.56 (± 2.75)	84.42 (± 0.25)	61.93 (± 2.35)	79.19 (± 0.39)
Core WL	**87.47** (± 1.08)	47.82 (± 4.62)	**85.01** (± 0.19)	59.43 (± 1.20)	79.24 (± 0.34)
PM	80.66 (± 0.90)	42.17 (± 2.02)	72.27 (± 0.59)	56.41 (± 1.45)	77.34 (± 0.97)
Core PM	**87.19** (± 1.47)	42.42 (± 1.06)	**74.90** (± 0.45)	**61.13** (± 1.44)	77.72 (± 0.71)

Method \ Dataset	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	REDDIT MULTI-12K
GR	59.85 (± 0.41)	35.28 (± 0.14)	76.82 (± 0.15)	35.32 (± 0.09)	22.68 (± 0.18)
Core GR	**69.91** (± 0.19)	**47.34** (± 0.84)	**80.67** (± 0.16)	**46.77** (± 0.09)	**32.41** (± 0.08)
SP	60.65 (± 0.34)	40.10 (± 0.71)	83.10 (± 0.22)	49.48 (± 0.14)	35.79 (± 0.09)
Core SP	**72.62** (± 0.59)	**49.43** (± 0.42)	**90.84** (± 0.14)	**54.35** (± 0.11)	**43.30** (± 0.04)
WL	72.44 (± 0.77)	51.19 (± 0.43)	74.99 (± 0.57)	49.69 (± 0.27)	33.44 (± 0.08)
Core WL	**74.02** (± 0.42)	51.35 (± 0.48)	**78.02** (± 0.23)	50.14 (± 0.21)	**35.23** (± 0.17)
PM	68.53 (± 0.61)	45.75 (± 0.66)	82.70 (± 0.68)	42.91 (± 0.42)	38.16 (± 0.19)
Core PM	**71.04** (± 0.64)	**48.30** (± 1.01)	**87.39** (± 0.55)	**50.63** (± 0.50)	**42.89** (± 0.14)

TABLE 5.1: Classification accuracy (± standard deviation) of the graphlet kernel (GR), shortest path kernel (SP), Weisfeiler-Lehman subtree kernel (WL), pyramid match kernel (PM) and their core variants on the 10 graph classification datasets. Core variants with statistically significant improvements over the base kernels are shown in bold as measured by a t-test with a p value of ≤ 0.05.

	MUTAG	ENZYMES	NCI1	PTC-MR	D&D	IMDB BINARY	IMDB MULTI	REDDIT BINARY	REDDIT MULTI-5K	REDDIT MULTI-12K
SP	1.69x	2.52x	1.62x	1.65x	3.00x	12.42x	17.34x	1.04x	1.05x	1.18x
GR	1.85x	2.94x	1.75x	1.50x	3.44x	7.95x	8.20x	2.24x	2.37x	2.80x
WL	1.76x	2.77x	1.68x	1.62x	3.34x	7.13x	6.84x	1.52x	1.58x	1.54x
PM	1.87x	2.79x	1.68x	1.50x	3.67x	6.92x	6.33x	1.90x	1.98x	1.96x
δ^*_{ave}	2.00	2.98	1.98	1.73	3.96	9.15	8.15	2.33	2.27	2.24

TABLE 5.2: Comparison of running times of base kernels vs their core variants. The values indicate the relative increase in running time when compared to the corresponding base kernel.

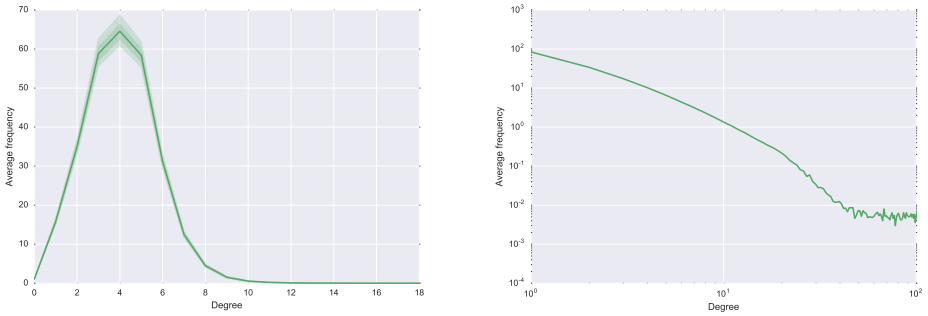


FIGURE 5.1: Degree distribution of D&D (left) and REDDIT-BINARY (right) datasets. Both axis of the right figure are logarithmic.

In terms of runtime, we can observe that in most cases, the extra computational cost required to compute the core variant of a kernel is negligible. We computed the average degeneracy δ^*_{ave} of the graphs contained in each dataset (shown in Table 5.2), and we observed that the running time is very related to its value. On the IMDB-BINARY and IMDB-MULTI datasets, computing the core variant requires more than 6 times the time of computing the base kernels. However, even that increase in running time is by no means prohibitive. It is also interesting to note that the extra computational cost comes with a significant improvement in accuracy.

We next investigate why the core variants lead to greater improvements on the social interaction datasets compared to the bioinformatics and chemoinformatics datasets. We attribute this difference in the behavior of the core variants to the underlying structure of the two types of graphs.

Figure 5.1 illustrates the degree distribution of the D&D and REDDIT-BINARY datasets. We observe that the latter follows the well-known power-law distri-

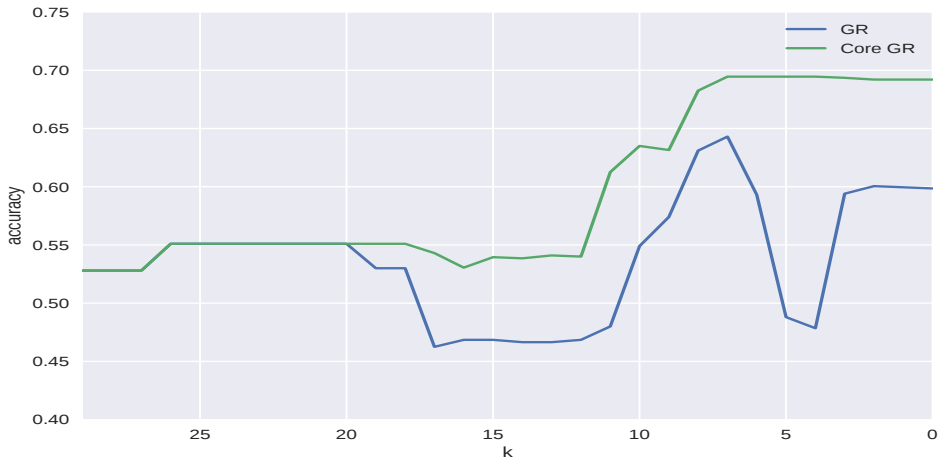


FIGURE 5.2: Classification accuracy of the graphlet kernel (GR) and its core variant (core GR) on the IMDB-BINARY dataset for the whole range of k-cores.

bution while the former does not. We should mention that we have observed almost identical behavior on the other bioinformatics/chemoinformatics and social interaction datasets, and the plots were omitted for illustration purposes. We can safely assume that the higher-order cores of the graphs of the REDDIT-BINARY dataset capture the most informative areas of the graph. Conversely, in graphs with structure similar to that of the graphs of the bioinformatics datasets, many nodes may end up sharing the exact same core number due to the coarse granularity of the k-core decomposition (leading to small degeneracies).

Finally, we compare the core GR kernel with its base kernel on the whole range of k-cores on the IMDB-BINARY dataset. For $k \in \{0, \ldots, 29\}$, we compute the GR kernel and its core variant, perform graph classification, and compare the achieved classification accuracies. The obtained results are shown in Figure 5.2. We can see that for $k < 20$, core GR systematically leads to better accuracies compared to its base kernel. The same behavior was also observed on most of the remaining datasets. An interesting observation is that for some k, by retaining only the internal k-cores of the graphs, we can get better classification accuracies compared to the 0-cores (i.e. the input graphs).

5.5 Conclusion

In this chapter, we defined a general framework for improving the performance of graph comparison algorithms. The proposed framework allows existing algorithms to compare structure in graphs at multiple different scales. The conducted experiments highlight the superiority in terms of accuracy of the core variants over their base kernels at the expense of only a slight increase in computational time. Moreover our method is available and implemented in the GraKel Library [Siglidis et al., 2020] as an easy plug and play method.

Finally, graph kernels capitalized greatly on the k-core decomposition, as every kernel on a different level takes advantage on comparing dense subgraphs. In this framework though, k-core decomposition was used as means of preprocessing in a way. The gain in performance though is ensured intuitively by the fact that the feature space induced by the core kernel variant contains the initial feature space of the original kernel.

As we want to go even further and see how the degeneracy framework could impact learning in deep learning, and try, as we did with core kernel, to input degeneracy as a valid and high performing tool in deep learning. It has already been used to scale auto-encoders for link prediction in graphs [Salha et al., 2019a] and we already experimented with graph metrics and algorithms to learn hidden set representations successfully with neural networks in [Skianis et al., 2020] using bipartite matching. But we want to provide an adaptation of the k-core decomposition to fit directly most of the neural network architectures.

CHAPITRE 6

Hcore-Init : Graph Degeneracy based Neural Network Initialization

In the previous chapter we applied successfully the k-core framework to graph kernels. Successfully indeed due to the accuracy improvement over the benchmark datasets at the expense of a slight increase in computational runtime. There, our framework can be characterized as a preprocessing method for graph kernels. Hence we want to go even further and try to apply k-core framework or variants to the learning method itself. As we can not do this directly to kernel SVM a natural candidate are Neural Network architectures.

Neural networks have become a very popular tool for many machine learning task, as in recent years we witnessed many novel architectures, learning and optimization techniques for deep learning. Capitalizing on the fact that neural networks inherently constitute multipartite graphs among neuron layers, we aim to analyze directly their structure to extract meaningful information that can improve the learning process. To our knowledge graph mining techniques for enhancing learning in neural networks have not been thoroughly investigated. In this chapter we propose an adapted version of the k-core structure for the complete weighted multipartite graph extracted from a deep learning architecture. As a multipartite graph is a combination of bipartite graphs, that are in

turn the incidence graphs of hypergraphs, we design k-hypercore decomposition, the hypergraph analogue of k-core degeneracy.

We applied k-hypercore to several neural network architectures, more specifically to convolutional neural networks and multilayer perceptrons for image recognition tasks after a very short pretraining. Then we used the information provided by the hypercore numbers of the neurons to re-initialize the weights of the neural network, thus biasing the gradient optimization scheme. Extensive experiments proved that k-hypercore outperforms the state-of-the-art initialization methods.

6.1 Introduction

During the last decade deep learning has been intensely in the focus of the research community. Its applications on a huge variety of scientific and industrial fields highlighted the need for new approaches at the level of neural network design. Researchers have studied until today different aspects of the Neural Network (NN) architectures and how these can be optimal for various tasks, i.e the optimization method used for the error backpropagation, the contribution of the activation functions between the NN layers or normalization techniques that encourage the loss convergence, i.e batch normalization, dropout layer, etc.

Weight initialization is one of the aspects of NN model design that contribute the most to the gradient flow of the hidden layer weights and by extension to the ability of the neural network to learn. The main focus on the matter of weight initialization ([Glorot and Bengio, 2010], [He et al., 2015]) is the observation that weights among different layers can have a high variance, making the gradients more likely to explode or vanish.

Neural Networks capitalize on graph structure by design (see Figure 6.1). Surprisingly there has been very few work analyzing them as a graph with edge and/or nodes attributes. Recent work [Skianis et al., 2020] introduces graph

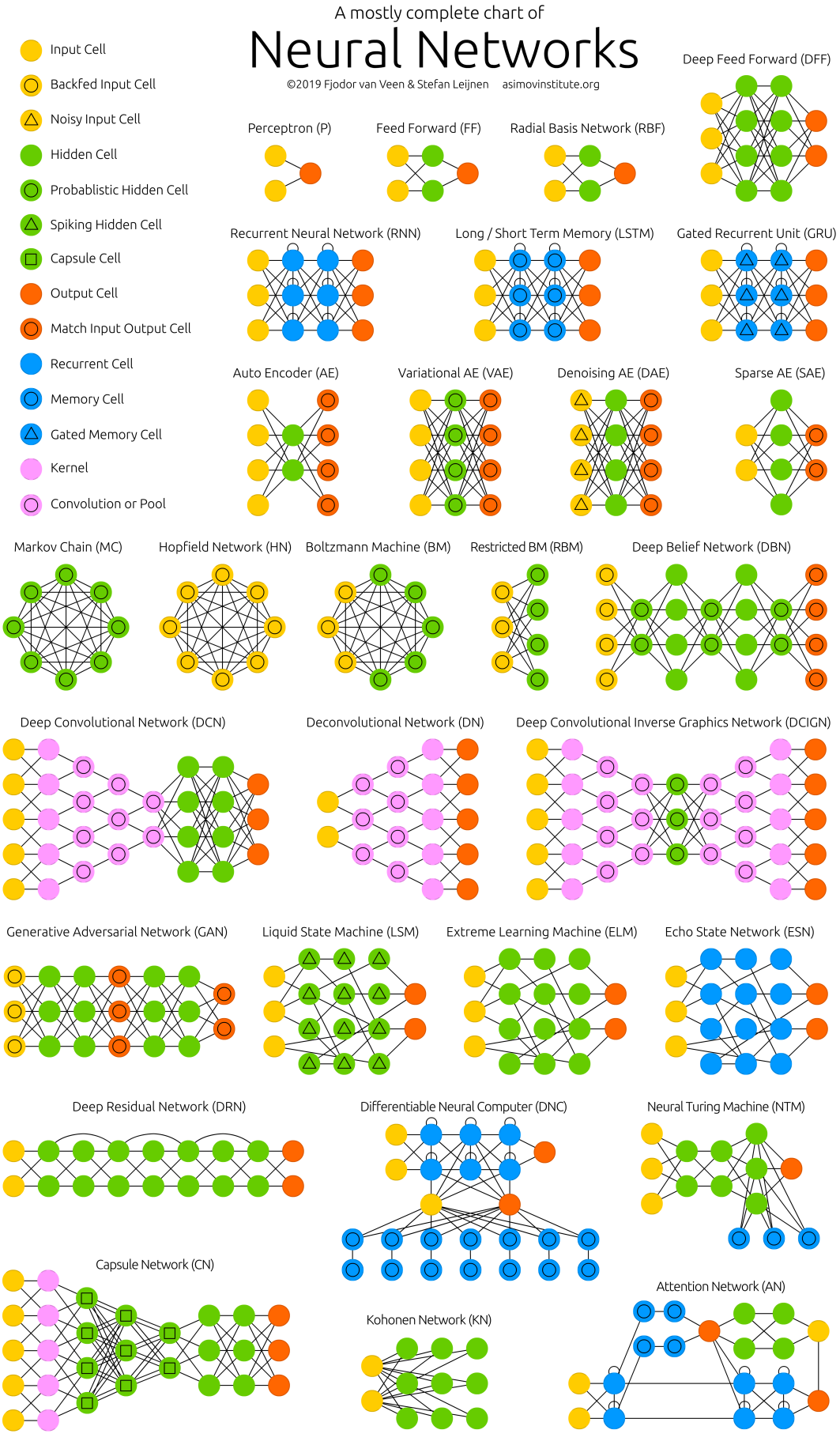


FIGURE 6.1: Simplified Neural Network representations as graphs (from https://www.asimovinstitute.org/author/fjodorvanveen/)

metrics to produce latent representation sets capitalising on bipartite matching directly implemented in the neural network architecture, which proved to be a very powerful method. Also the work by C. Morris analyzes the expressivity of Graph Neural Networks using the Weisfeiler-Leman isomorphism test [Morris et al., 2019]. Our interest lies in trying to refine the optimization scheme by capitalizing on graph metrics and decompositions. One natural candidate was the k-core decomposition [Seidman, 1983a]. Indeed this decomposition method, being very efficient ($O(n\log(n))$ in the best cases [Batagelj and Zaversnik, 2003]), performs very well in state-of-the-art frameworks for enhancing supervised learning methods [Nikolentzos et al.,]. Providing key subgraphs, and also extracting very good features.

Unfortunately, in the case of a graph representing a neural network, k-core might lack some features. As a matter of fact, graphs extracted from NNs constitute multipartite complete weighted graphs, in the case of an Multilayer Perceptron and almost complete for Convolutional Neural Networks. As we saw different k-core variants for different types of graphs, such as the k-truss [Rossi et al., 2015] counting triangles, the D-core [Giatsidis et al., 2013] for directed graphs, were designed this past decade. A natural thought was then to design our own version of the k-core for our precise graph structure.

Hence our contributions are the following :

— We provide a unified method of constructing the graph representation of a neural network as a block composition of the given architecture (see Figure 6.3). This is achieved by transforming each part of the network (i.e linear or convolutional layers, normalization/dropout layers and pooling operators) into a subgraph. Having this graph representation, it is possible to apply different types of combinatorial algorithms to extract information from the graph structure of the network.

— Next we design a new degeneracy framework, namely the **k-hypercore**, extending the concept of k-core to bipartite graphs by considering that each pair of layers of the neural network, constituting a bipartite graph, is

the incidence graph of a hypergraph (see Figure 6.2).

— we propose a novel weight initialization scheme, **Hcore-init** by using the information provided by the weighted version of the k**-hypercore** of a NN extracted graph, to re-initialize the weights of the given neural network, in our case, a Convolutional neural network and a Multilayer Perceptron. Our proposal clearly outperforms traditional initialization methods on classical deep learning tasks.

The rest of this chapter is organized as follows, first some preliminary definitions and overview of the state of the art methods in neural network initialization methods. Then we provide the methodology which allows us to transform neural networks to edge weighted graphs. Further on, we proceed to the main contribution of the chapter being the definition of the hypercore degeneracy and the procedure which produces our initialization method. Finally we test our method on several image classification datasets, comparing it the main initialization method used in neural networks.

6.2 Preliminaries

In deep neural networks, weight initialization is a vital factor of the performance of different architectures [Mishkin and Matas, 2015]. The reason is that an appropriate initialization of the weights of the neural network can avert the explosion or vanishing of the layer activation output values.

6.2.1 Initialization Methods

Glorot Initialization

One of the most popular initialization methods is Glorot initialization [Glorot and Bengio, 2010]. According to that, the weights of the network are initialized by drawing them from a normal distribution with $E[W] = 0$ and $\mathrm{Var}(w_i) = 1/\mathrm{fanin}$, where fanin is the number of incoming neurons. Also, more generally, we can define variance with respect to the number of outgoing neurons as : $\mathrm{Var}(w_i) = 1/(\mathrm{fanin} + \mathrm{fanout})$, where fanout is the number of neurons that the output is directed to.

He Initialization

Although Glorot initialization method manages to maintain the variance of all layers equal, it assumes that the activation function is linear. In most of the cases of non-linear activation function that Glorot initialization is used, the hyperbolic tangent activation is employed. The need for taking into account the activation function for the weight initialization led to the He Initialization [He et al., 2015]. According to this method, in the case that we employ *ReLU* activation functions, we initialize the network weights by drawing samples from a normal distribution with zero mean : $E[W] = 0$ and variance that depends on the order of the layer : $\text{Var}[W] = 2/(n^l)$, where l is the index of the l-th layer and n the number of neurons in the given layer.

One main assumption for weight initialization is that the mean of the random distribution used for initialization needs to be 0. Otherwise, the calculation of the variances presented above could not be done and we won't be able to have a fixed way to initialize the variance.

Since in our work we want to capitalize on the k-hypercore decomposition to *bias* those distributions we will have to face the fact that we might not be able to control the variance of the weights we initialize. Thankfully the fact that the initial distribution has 0 mean will ensure that our method respects as well this condition on every layer of the neural network.

Moreover, since the k-hypercore decomposition is defined over hypergraphs, let us remind some properties of hypergraphs and its relations with bipartite graph.

6.2.2 Hypergraphs and Bipartite Graphs

A hypergraph is a generalization of graph in which an edge can join any number of vertices. It can be represented and we keep this notation for the rest of the chapter as $\mathcal{H} = (V, E_{\mathcal{H}})$ where V is the set of nodes, and $E_{\mathcal{H}}$ is the set of hyperedges, i.e. a set of subsets of V. Therefore $E_{\mathcal{H}}$ is a subset of $\mathcal{P}(V)$.

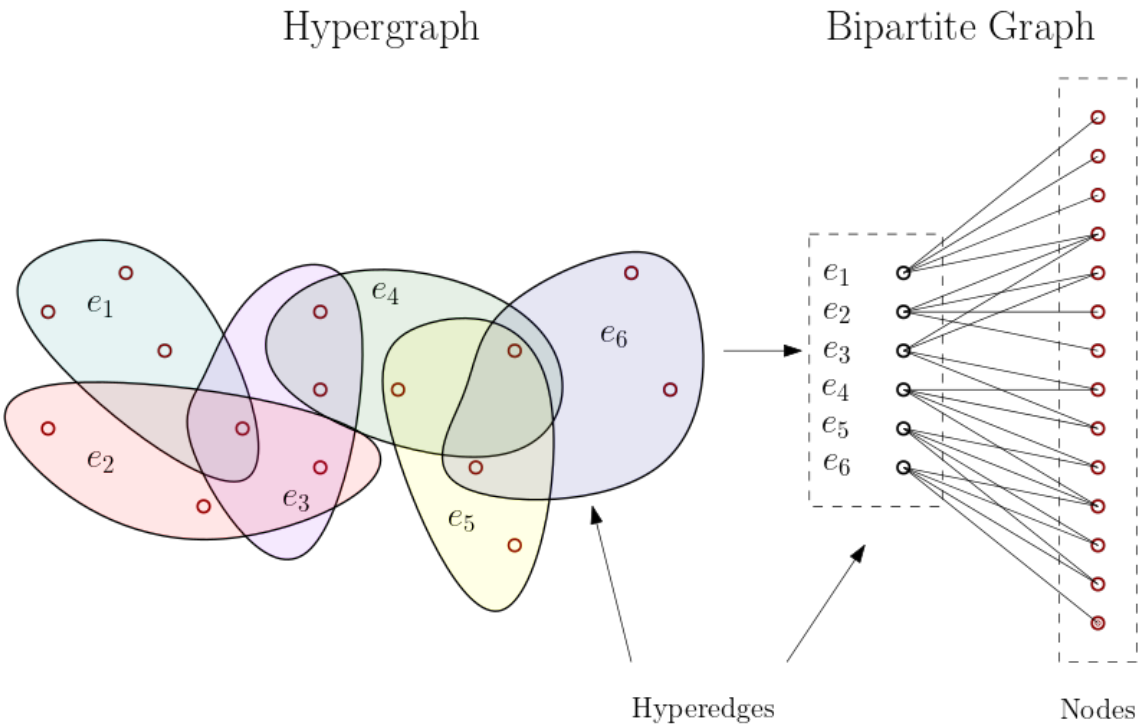


FIGURE 6.2: Hypergraph and the corresponding incidence graph

Moreover a bipartite graph is the incidence graph of a hypergraph [Pisanski and Randic, 2000]. Indeed, a hypergraph $\mathcal{H}$ may be represented by a bipartite graph $\mathcal{G}$ as follows : the sets X and E are the partitions of $\mathcal{G}$, and (x_1, e_1) are connected with an edge if and only if vertex x_1 is contained in edge e_1 in $\mathcal{H}$. Conversely, any bipartite graph with fixed parts and no unconnected nodes in the second part represents some hypergraph in the manner described above. Hence, we can consider that every pair of layers in the neural network can be viewed as a hypergraph, where the left layer represent the hyperedges and the right the nodes (see Figure 6.2).

6.3 Graph Extraction from Neural Network Architecture

We will now describe how we map the two classic neural network architectures we investigate to graphs, and more specifically to a collection of bipartite ones. Also, from now on, we are going to refer to a fully-connected neural net-

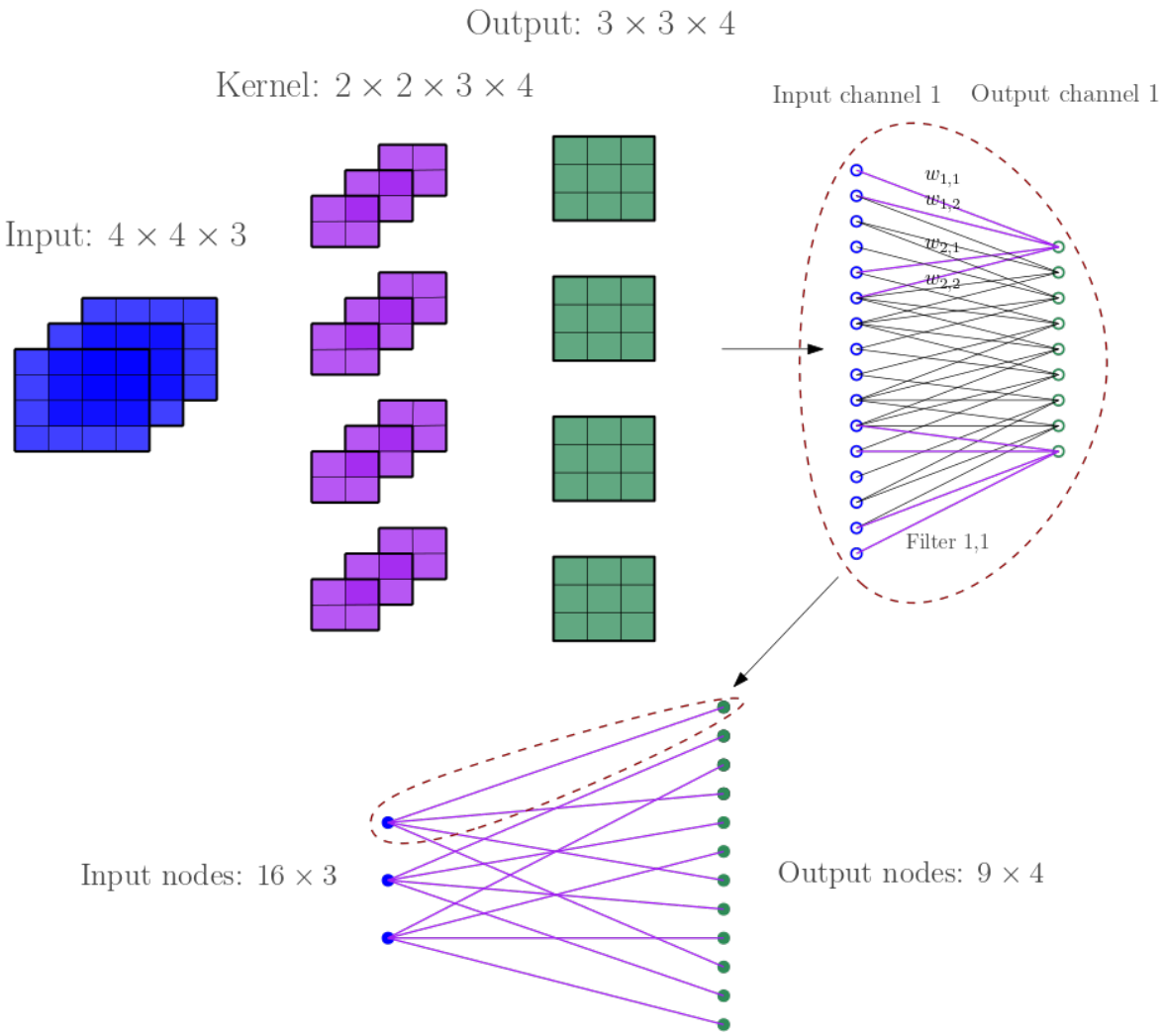


FIGURE 6.3: Illustration of the transformation of a CNN to graph.

work as FCNN and to a convolutional neural network as CNN [Krizhevsky et al., 2012].

6.3.1 Fully-Connected Neural Networks

Let a FCNN $\mathcal{F}$ with L hidden layers, n_i, $i = 1, .., L$ number of hidden units per layer and $W_i \in \mathbb{R}^{n_i, n_{i+1}}$ the weight matrix of the links between the units of the layers i and $i + 1$.

We define the graph $G_{\mathcal{F}} = (V, E, W)$ as the graph representation of the FCNN $\mathcal{F}$, where the set of nodes V corresponds to the $\sum_{i=1}^{L} n_i$ number of hidden units of $\mathcal{F}$, the set of edges E contains all the links of unit pairs across the layers of $\mathcal{F}$ and the edge weight matrix W corresponds to the link weight matrices $W_i, i = 1, ..., L-1$. We note that the graph representation $G_{\mathcal{F}}$ does not take into account any activation functions σ used in $\mathcal{F}$.

Remark. It is easy to see that G_F is a k-partite graph (i.e a graph whose vertices can be partitioned into k independent sets) and more specifically a union of $L - 1$ complete bipartite graphs.

6.3.2 Convolutional Neural Networks

After showing the correspondence between a FCNN $\mathcal{F}$ and its graph representation $G_{\mathcal{F}}$, we are ready to define the graph representation of a CNN layer. Let a CNN layer $\mathcal{C}$. The convolutional layer is characterized by the input information that has I input channels where each channel provides $n \times n$ features (i.e an 24×24 image characterized by the 3 RGB channels), the output information that has O output channels, where each channel has $m \times m$ features and the matrix of the convolutional kernel $F \in \mathbb{R}^{w \times h \times I \times O}$, where w, h are the width and height of the kernel.

In order to define the graph $G_{\mathcal{C}} = (V, E, W)$ as the graph representation of the CNN $\mathcal{C}$, we have to flatten the 3 and 4-dimensional input, output, and filter matrices correspondingly. Specifically, the $G_{\mathcal{C}}$ is a bipartite graph, where the

first partition of nodes P_1 is the flattened input information of the CNN layer ($|P_1| = I \times n \times n$) , the second partition of nodes P_2 is the flatten output information ($|P_2| = O \times m \times m$).

6.4 Weight Initialization Based on Hcore

As degeneracy frameworks have proven to be very efficient at extracting influential individuals in a network [Al-garadi et al., 2017], we are motivated to consider structural information provided by the hcore decomposition of the network to identify "influential" neurons.

Assuming a neural network graph, we provide a definition of degeneracy specifically for hypergraphs, where standard k-core does not apply.

Definition 6.4.1 (Hypercore)**.** Given a hypergraph $\mathcal{H} = (V, E_{\mathcal{H}})$ We define the (k, l)-**hypercore** as a maximal connected subgraph of $\mathcal{H}$ in which all vertices have hyperdegree at least k and all hyperedges have at least l incident nodes.

As for now on, we will refer to the $(k, 2)$-hypercore as the k-hcore and similarly, the *hcore number* of the node will be the largest value of k for which the given node belongs to the k-hcore.

This provides a hypergraph decomposition and in our case a decomposition of the right handside of the studied bipartite graph (see Figure 6.4), as we do not care about the hcore of the hyperedges.

Since we deal with edge-weighted bipartite graphs, we will use the weighted degree to define the hcore ranking of the nodes given the following weighted-hypercore definition.

Definition 6.4.2 (Weighted-hypercore)**.** Given an edge weighted hypergraph $\mathcal{H} = (V, E_{\mathcal{H}})$, we define the (k, l)-**weighted-hypercore** as a maximal connected subgraph of $\mathcal{H}$ in which all vertices have hyper-weighted-degree at least k and all hyperedges have at least l incident nodes.

Again, we will refer to the $(k, 2)$-weighted-hypercore as k-WHcore. Now that we have this weighted version, we need to define a way to initialize the weights

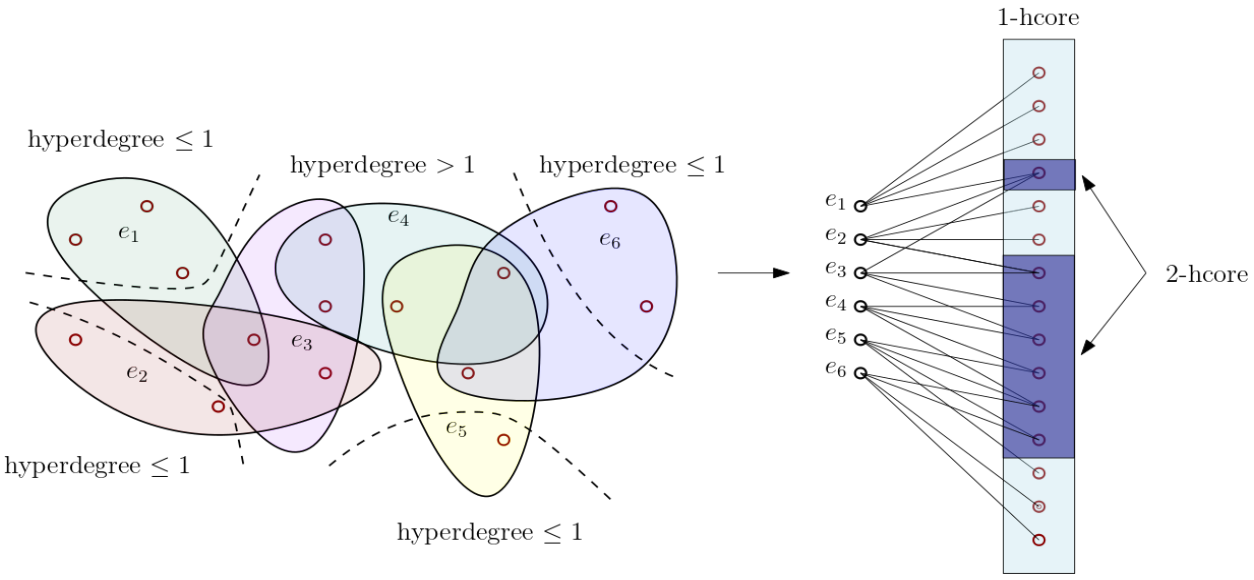


FIGURE 6.4: Example of a k-hcore decomposition of a hypergraph

Algorithm 6 Hcore decomposition algorithm

```
procedure HCORE(G, rnodes)
    Input G : bipartite graph, rnodes : right layer nodes
    Output hcore : dictionary of hcore values

    hcore ← dict((node, 0) for node in rnodes)
    tokeep ← rnodes
    while tokeep ≠ ∅ do
        state ← True
        while state == True do
            state ← False
            tokeep ← []
            for node ∈ rnodes do
                if G.degree(node) > k then
                    tokeep.append(node)
                else
                    hcore[node] = k
                    graph.remove[node]
                    state ← True
                end if
            end for
            for node ∈ G.nodes \ rnodes do
                if G.degree(node) = 1 then
                    G.remove(node)
                end if
            end for
        end while
        k ← k + 1
    end while
end procedure
```

of the neural network. Indeed, since the WHcore is a value given to the nodes of the network and not the edges, being the weights we aim to initialize. The WHcore shows us which neurons gather the more information, positive on the one hand and negative on the other. After a quick pretraining, we learn the weights just enough to show which neurons have a higher impact on the learning. This information is then grouped by the WHcore into influential neurons and less influential ones.

Moreover, since weights in neural networks are sampled from centered normal law, we have positive and negative weights. Since the WHcore framework operates on positive weighted degrees, we provide two graph representations of the neural network, namely G^+ and G^-. The G^+ graph is built upon the positive weights of the neural network, and the edge weights of the G^- graph are the absolute values of the negative weights of the neural network. Indeed if between neuron x_i and neuron y_j, $w_{ij} > 0$ then we add an edge with weight w_{ij} between node x_i and y_j in graph G^+, otherwise we add an edge with weight $|w_{ij}|$ between node x_i and y_j to graph G^-.

Remark. It is important to note that the **WHcore number** of a node is the largest k in which a node is contained in the k-WHcore. Using the notations from 4 the WHcore number of a node is a function whose definition is based on vertex degree. It is indeed possible that two vertices of the same degree end up to different core levels. The value of the parameter depends on the graph structure, and in particular on the interconnectivity of high degree vertices, and the corresponding decomposition is defined using vertex layouts and the notations from chapter 4. Hence defining the Hcore of a graph $\delta(\mathcal{H})$ is defined formally as follows.

$$\delta(\mathcal{H}) = \min_{\forall L} \max_{\substack{v_i \in L \\ i \in \{1,\dots,n\}}} \mu(v_i, L_{\leq i}), \tag{6.1}$$

where the $\mu(v_i, L_{\leq i})$ function is the degree contribution of v_i in $L_{\leq i}$, where the layouts are defined over the right side nodes in the bipartite graph (on the

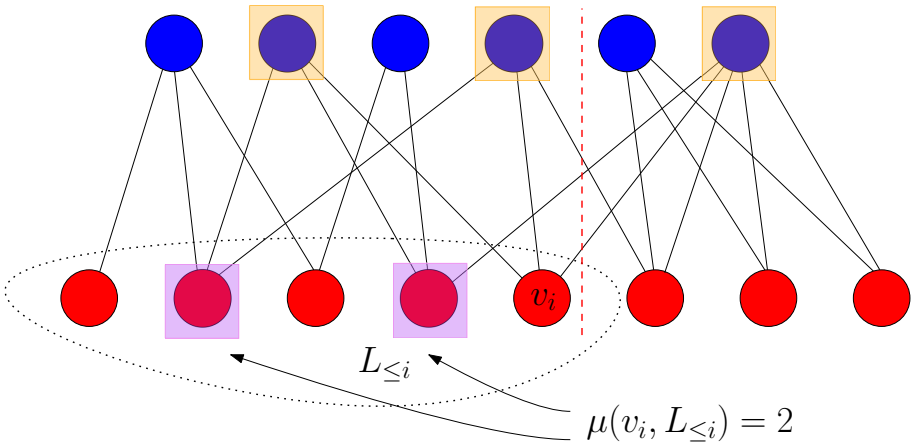


FIGURE 6.5: degree of the degree of v_i in the given layout, note that the layout does not concern top nodes (hyperedges).

bottom in figure 6.5). Observe that the vertex v_i has degree 3 while its degree contribution to the depicted layout of the red vertices is 2.

In order to simplify those notations for the rest of this chapter, as the degree depends on the weights, there exists two functions g and h such that $g(W, x)$ outputs the weighted degree of a node x, thus being a linear combination of the weights W. Then $c(W, x) = h(g(W, x))$ is the WHcore number of the node x. For convenience, we now write $c(W^+, x_k) = c_k^+$ where W_k are the positive weights of the weight matrix W.
Moreover, the following initialization schemes are done after a small amount of pretraining of the neural network, in order to have a preliminary information over the importance on the task of the neurons.

6.4.1 Initialization of the FCNN

The initialization then is then dependent on the architecture we are looking at, indeed for an FCNN as the graph construction is fairly straightforward we proceed as follows.

For every pair of layers for both positive and negative graphs, we have nodes x_i, with $i \in \{1, \ldots,$ fanin$\}$ in the left side of the bipartite graph, and y_j nodes, with $j \in \{1, \ldots,$ fanout$\}$ nodes on the right side. As for every node y_i we compute

their WHcore from the graph G^-, c_j^- and from the graph G^+, c_j^+. Then the given layer weights $w_{i,j}$, are initialized, depending on their sign, with a normal law with expectancy :

— for all i if $w_{i,j} \geq 0$, $M = \frac{c_j^+}{\sum_{1 \leq k \leq \text{fanout}} c_k^+}$,

— else $M = \frac{c_j^-}{\sum_{1 \leq k \leq \text{fanout}} c_k^-}$

and with the same variance used in He initialization. We prove later that the overall mean value of the new random variable obtained in this fashion is 0 as well, justifying the use of the He variance to be optimal.

6.4.2 Initialization of the CNN

For the CNN, since the induced graph is more intricate and the filter weights must follow the same distribution, the initialization framework has to be adapted. We still compute the WHcore on a pair of layers but keeping the filters in mind, the left layer nodes are $x_i^{(k)}$ with $i \in \{1, \dots, n \times n\}$ the input size and $k \in \{1, \dots, I\}$ the number of input filters. Similarly the left layer nodes are $y_j^{(k')}$, where $j \in \{1, \dots, m \times m\}$ the output size, and $k' \in \{1, \dots, O\}$ the output channels. We remind as well that we have two WHcores, one for the positive graph c^+ and one for then negative c^-. Then for a given filter $w^{(k,k')}$ its values are initialized with the following method :

— we define f for a given filter W as $m(W^+) = \frac{1}{H^2} \sum_j c_j^+$ and $m(W^-) = \frac{1}{H^2} \sum_j c_j^-$, if $m(W^+) - m(W^+) > 0$ then $M = m(W^+)$

— else $M = -m(W^-)$.

Using the notations given in the previous remark we can write m in the following general form :

$$m = \text{sign}(\text{argmax}(m(W^+), f(W^-))) \max(m(W^+), m(W^-))$$

where $\text{sign}(W^+) = 1$ and $\text{sign}(W^-) = -1$.

This initialization is done for every filter and with variance given by the He initialization method. Now we will prove that for the CNN the overall expectancy of the mean value produced is indeed 0.

Proposition 6.4.3. *Let X_1 and X_2 two centered i.i.d. random variables with symmetric distribution. We define $X^+ = max\{X_1, 0\}$, $X^- = max\{X_2, 0\}$, and a real valued measurable function $f : \mathbb{R}_+ \to \mathbb{R}$ such that $\mathbb{E}[|f(X^+)|] < \infty$ and $\mathbb{E}[|f(X^-)|] < \infty$.*

Then :

— *X^+, X^- are positive i.i.d. random variables.*

— *The random variable :*

$$Z = sign\big(argmax(f(X^+), f(X^-))\big) max\big(f(X^+), f(X^-)\big)$$

is centered, i.e. $\mathbb{E}[Z] = 0$

Démonstration. We remind that the function $\mathbb{I}_{\{x \in X\}}$ is the Euler indicator function :

$$\mathbb{I}_{\{x \in X\}} = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{otherwise.} \end{cases}$$

Let us proceed to evaluate the expectancy of Z provided that X^+ and X^- are i.i.d. :

$$\begin{aligned} &\mathbb{E}[Z] = \mathbb{E}[Z\mathbb{I}_{\{f(X^+)>f(X^-)\}}] + \mathbb{E}[Z\mathbb{I}_{\{f(X^+)\leq f(X^-)\}}] = \\ &\mathbb{E}[f(X^+)\mathbb{I}_{\{f(X^+)>f(X^-)\}}] - \mathbb{E}[f(X^-)\mathbb{I}_{\{f(X^+)\leq f(X^-)\}}] = \\ &\mathbb{E}[(f(X^+) + f(X^-))\mathbb{I}_{\{f(X^+)>f(X^-)\}}] - \mathbb{E}[f(X^-)]. \end{aligned}$$

Given the initial assumptions , we can expand the first term $\mathbb{E}[f(X^+)\mathbb{I}_{\{f(X^+)>f(X^-)\}}]$ as follows :

$$\mathbb{E}[f(X^+)\mathbb{I}_{\{f(X^+)>f(X^-)\}}] = \iint f(X^+)\mathbb{I}_{\{f(X^+)>f(X^-)\}} dP(X^+)dP(X^-)$$

As X^+ and X^- follow the same distribution, and f is a measurable function, we use the *Fubini* theorem to intervert the integrals as follows :

$$\mathbb{E}[f(X^+)\mathbb{I}_{\{f(X^+)>f(X^-)\}}] = \iint f(X^-)\mathbb{I}_{\{f(X^-)\geq f(X^+)\}}dP(X^-)dP(X^+).$$

Now replacing this in the original equation gives us :

$$\begin{aligned}
\mathbb{E}[Z] &= \iint f(X^-)\mathbb{I}_{\{f(X^-)\geq f(X^+)\}}dP(X^-)dP(X^+) \\
&\quad + \iint f(X^-)\mathbb{I}_{\{f(X^+)>f(X^-)\}}dP(X^-)dP(X^+) \\
&\quad -\mathbb{E}[f(X^-)] \\
&= \mathbb{E}[(f(X^-)\mathbb{I}_{\{f(X^-)\geq f(X^+)\}}] \\
&\quad +\mathbb{E}[(f(X^-)\mathbb{I}_{\{f(X^+)>f(X^-)\}}] - \mathbb{E}[f(X^-)] \\
&= 0
\end{aligned}$$

This completes our proof that Z is a centered random variable. □

Notice that setting the function $m = l \circ g \circ h$ we can write $l \circ g = f$ and $X = h(W)$. As we defined previously h to be the weighted degree function of a node :

$$h(W_j^+) = \sum_i W_{ij}\mathbb{I}_{\{W_{ij}>0\}}$$

$$h(W_j^-) = \sum_i |W_{ij}|\mathbb{I}_{\{W_{ij}\leq 0\}}$$

which ensures that $h(W^+)$ and $h(W^-)$ follow the same distribution by linear combination of absolute value of the same normal distribution. Replacing these function in the previous proposition, i.e. $f = l \circ g$, $X^+ = h(W^+)$ and $X^- = h(W^-)$ proves that our initialization method has mean 0. This proof allows us to justify the use of the He variance in our initialization method as it was proven to be the optimal one.

6.5 Experiments

We will now evaluate our proposed weight initialization method Hcore-Init in image classification task using three standard datasets, CIFAR-10, CIFAR-100, and MNIST. We compare Hcore-Init to the results of He initialization scheme. It is important to stress that we do not experiment on state-of-the-art architectures for each dataset. We want to show, as our method can be used separately on different architecture blocks, i.e. only at the convolutional layers, or only at the FCNN part, or both. We observe that it outperforms standard initialization methods, regardless of the block of the architecture that is initialized. Hence in this section, we evaluate image classification accuracy on the aforementioned datasets with simple CNN architectures presented in this section.

6.5.1 Dataset Specifications

The CIFAR-10 and CIFAR-100 datasets are labeled subsets of the 80 million tiny images dataset collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton [Krizhevsky et al., 2009].

— The CIFAR-10 dataset consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

— The CIFAR-100 is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class.

We also test our model on the MNIST database of handwritten digits, providing a training set of 60000 examples, and a test set of 10000 examples. The digits have been size-normalized and centered in a fixed-size image [LeCun and Cortes, 2010].

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than

another. Between them, the training batches contain exactly 5000 images from each class.

6.5.2 Model Setup and Baseline

Next, we present the models that were trained and evaluated for the image classification task. We note that for every case, we compare two scenarios :

1. Initialization of the model with He initialization [He et al., 2015], training on the train set for 150 epochs and evaluation on the test set.
2. Pretraining of the model (using He initialization) for N epochs, re-initialization of the model with Hcore-Init, training on the train set for the rest $150 - N$ epochs and evaluation on the test set. N has been set as a hyper-parameter.

For the CIFAR-10 and CIFAR-100 datasets, we applied 2 convolutional layers with sizes $3 \times 6 \times 5$ and $6 \times 15 \times 5$ respectively, where 5 is the kernel size and the stride was set to 1. Moreover, after each convolutional layer, we applied two 2×2 max-pooling operators and finally three fully connected layers with corresponding sizes 400×120, 120×84, $84 \times$ #classes, where #classes $=$ 10 and 100 respectively for the two datasets. Furthermore, we used *ReLU* as activation function among the linear layers and *tanh* for the convolution layers.

For the MNIST dataset, we applied again 2 convolutional layers of size $1 \times 10 \times 5$ and $10 \times 20 \times 5$, where again the filter size was set to 5 and the stride was set to 1. As in the other datasets, we employed two 2×2 max-pooling operators and we performed dropout [Srivastava et al., 2014] on the output of the 2^{nd} convolutional layer with probability $p = 0.5$. Finally, we applied 2 fully connected layers of size 320×50 and 50×10 and *ReLU* as an activation function throughout the layers.

In all cases, we employed stochastic gradient descent [Kiefer and Wolfowitz, 1952] with momentum set to 0.9 and learning rate set to 0.001. As we mentioned before, we chose 2 rather simple models, as we intend to highlight the contribution of Hcore-Init in comparison to its competitor and not to achieve state-of-the-art results for the given datasets, which are exhaustively examined.

6.5.3 Settings of the Weight Initialization

Next, we present the contribution of Hcore-Init to the performance of the neural network architecture with respect to its application on different types of layers. Specifically, we applied the configurations of the initialization methods (a) exclusively on the set of the linear layers (b) exclusively on the set of the convolutional layers (c) on the combined set of linear and convolutional layers of the model.

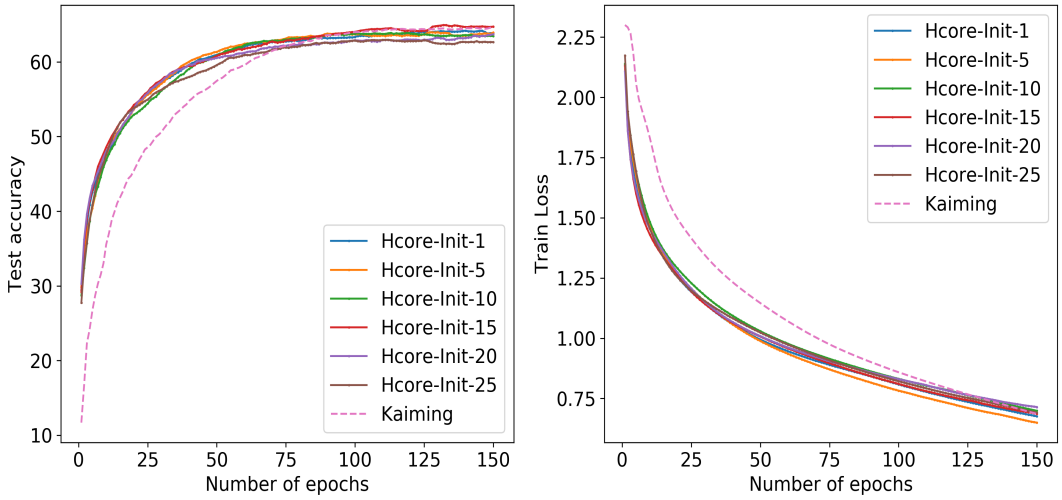


FIGURE 6.6: Test accuracy (left) and train loss (right) on CIFAR-10 for the combined application of the initialization on the linear and the convolutional layers. For the curves Hcore-init-x, x stands for the number of pretraining epochs.

On Figure 6.6, we observe that for 15 pretraining epochs, the model initialized with Hcore-Init outperforms the model initialized with He initialization. It is, also, noteworthy that the loss convergences faster when applying Hcore-Init. This highlights empirically our initial motivation of encouraging the "important" weights by using the graph information from the model architecture.

On Figure 6.7 and Figure 6.8, we can notice the contribution again of Hcore-Init in the performance of the network, when the former is applied on the fully connected and convolutional layers respectively. We can see that in both cases, Hcore-Init with different numbers of pretraining epochs (10 and 20 correspondly) achieves better accuracy results in comparison to He.

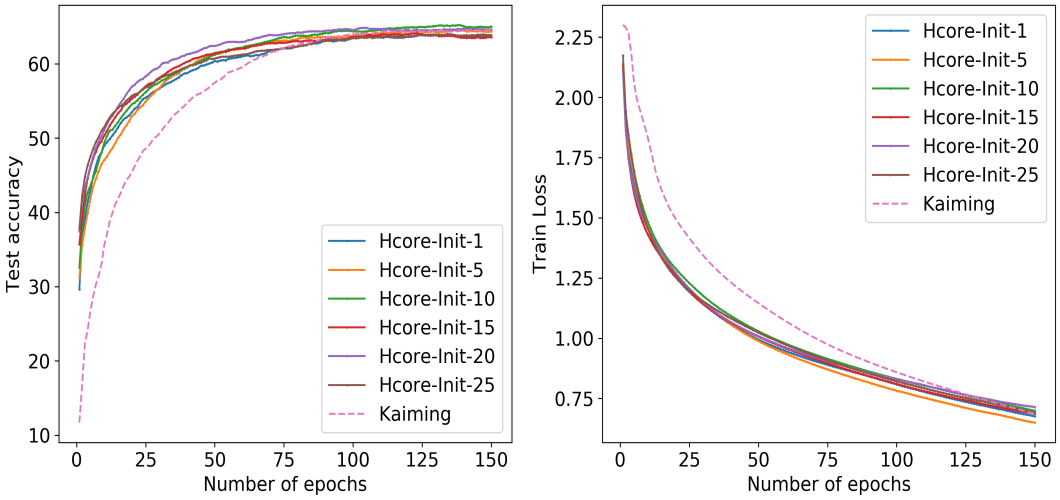


FIGURE 6.7: Test accuracy and train loss on CIFAR-10 for the initialization applied **only** on the linear layers.

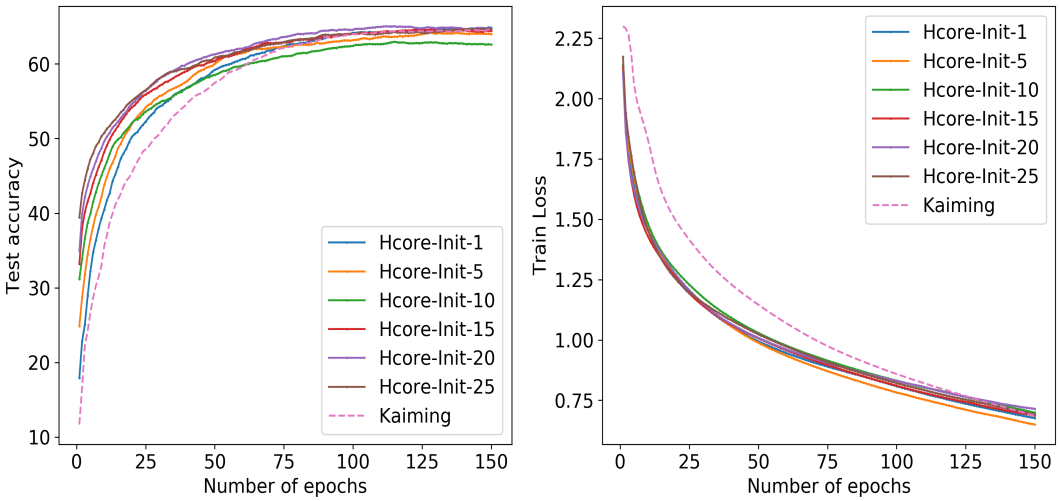


FIGURE 6.8: Test accuracy and train loss on CIFAR-10 for the initialization applied **only** on the convolutional layers.

Finally, we report the results of the experiments conducted on the 3 datasets in Table 6.1. Those results correspond to an ablation study over the different number of pretraining epochs as well as the different initialization scenarios, i.e. initializing only on the linear layers, convolutional layers, and the whole architecture. We kept for each mentioned scenario the best performance, and as

TABLE 6.1: Top Accuracy results over initializing the full model, only the CNN and only the FCNN for CIFAR-10, CIFAR-100, and MNIST. **Hcore-Init*** represent the top performance over all the pretraining epochs configurations up to 25

	CIFAR-10	**CIFAR-100**	**MNIST**
He	64.62	32.56	98, 71
Hcore-Init*	**65.22**	**33.48**	**98.91**
Hcore-Init-1	64.91	32.87	98.59
Hcore-Init-5	64.41	32.96	98.70
Hcore-Init-10	**65.22**	33.41	98.81
Hcore-Init-15	64.94	33.45	98.64
Hcore-Init-20	65.05	33.39	98.87
Hcore-Init-25	64.72	**33.48**	**98.91**

it is evident **Hcore-Init*** achieves the best overall accuracy. It is important to stress that we do not necessarily need a long pretraining phase to achieve the best results, in fact, only 10 epochs is usually more than enough to outperform in a significant way the He initialization. We remind that this pretraining corresponds to less than 10% of the total training which is proportional, in terms of computation time, to 10% of the time to train the model. Furthermore it is interesting to notice that in the early stages of pretraining we are more likely to lose some accuracy as the gradient direction in this stage of the training might be wrong. This justifies as well the consistency of our method.

6.6 Conclusion

In this chapter, we propose **Hcore-Init**, a novel initialization method applicable on the most common blocks of neural network architectures, i.e. convolutional and linear layers. This method capitalizes on a graph representation of the neural network and more specifically on the densest parts of it found by the hypergraph degeneracy methods we define, providing thus a neuron ranking for the bipartite architecture of the neural network layers. Our method, learning with a small pretraining of the neural network, outperforms the state of the art He initialization, under the condition that the initialization distribution has zero expectancy. We also see that, as the sensible parameter of our method is the number of pretraining epochs, taking 10 pretraining epochs is enough to have outperform the initial model in all cases. We also see that with a small number

of pretraining epochs our model loses accuracy over the He initialization method, which is also expected as the gradient descend in the early stages of the training process they can provide wrong gradient directions and value neurons that should not. This work is intended to be used as a framework to initialize specific blocks in more complex architectures that might bear more information and are more valuable for the task at hand.

CHAPITRE 7

Conclusion

In this dissertation, we saw that degeneracy is a versatile tool, used and refined by the graph learning community this past decade. With this in mind, our research focus is twofold, and built upon graph degeneracy, through the following motivations :

— Provide structural characterizations of edge-type degeneracies in order to understand them and have a clearer view on computational complexity for potential applications.

— Integrate degeneracy frameworks within learning algorithms and especially within deep learning architectures.

Both of these objectives require a good understanding in graph theory and machine learning frameworks, as well as graph mining techniques. These domains made this work even more exciting and lead to an interdisciplinary work, as most of the research done in general in Machine Learning. In the following sections we provide an overview of the *contributions* and we discuss future research directions.

7.1 Summary of Contributions

The contributions of the thesis can be summarized as follows.

Computational complexity argument for the s-edge-degeneracy. We proved in Chapter 3 that, as for the s-degeneracy can be computed in polynomial for all path lengths s in $\{1, 2, 3, 4, \infty\}$. Also, the s-edge-degeneracy can be as well computed in polynomial time for s in $\{1, 2, \infty\}$. It is important to remind that for the other values of s, the problem, and it is the case for the vertex degeneracy as well, is NP-complete.

Structural theorem for edge-admissibility. In Chapter 3 we also were able to position the proven result toward the existing structural theorems concerning clique exclusion and bounded admissibility. And, that enabled us to decompose a graph in almost-bounded-degree graphs. Additionally a main result was to provide an equivalent result to [Dvořák, 2012] for the edge-admissibility. Indeed, every graph with bounded edge-admissibility can be decomposed in edge-sums of almost-bounded-edge-degree graphs.

Unification of the degree degeneracy, connectivity degeneracy and admissibility framework. In Chapter 4 we gathered degree and connectivity degeneracy hierarchies as well as the corresponding admissibility hierarchies. We proposed a general framework based on four metrics adapted to the degree and connectivity frameworks, such that every one of those graph metrics can output a given hierarchy, throughout a min-max procedure. Furthermore, using results from Chapter 3, we were able to provide order relationships, first between degeneracy and corresponding admissibility, next between connectivity degeneracy and degree degeneracy.

Efficient algorithm for edge-connectivity degeneracy. We capitalize mainly on the previous contribution, especially on the orderings provided between connectivity and degree degeneracy, and on the contraction theorem of Menger. In the second part of Chapter 4 we designed an efficient algorithm to compute the edge-connectivity degeneracy decomposition. Unfortunately, even if the algorithm runs with a complexity of $\mathcal{O}(\frac{k}{2}n_{k/2}^2 \log^2(n_{k/2}))$, in the worst case scenario where $n_{k/2}$ is way smaller than n. We saw that in real world datasets, λ^*-edge-connectivity core was very close if not the same with the δ^*-core. This motivated us for the rest of the dissertation to use only degree-degeneracy.

k-core framework for graph kernels and performance criterion. In Chapter 5 we proposed a framework for including k-core decomposition into the graph kernel routine. Indeed, the proposed method is based on applying the k-core decomposition algorithm to the graphs in the dataset we aim to classify. Then, on the obtained subgraph-based structure, we were able to apply the chosen kernel via this decomposition. Once this is done, we sum the kernel matrices obtained for each core to form the final core-kernel. Finally, this framework outperformed on almost every situation its classic kernel analogue, at the expense of only a slight increase in computational time.

Hypergraph degree degeneracy definition and algorithm design. In the last part of this dissertation the contributions are twofold. First, in Chapter 6, we define a framework for degeneracy on hypergraphs. Not only this definition is novel, but is we use it to find core decompositions for bipartite graphs, as incidence graphs of hypergraphs. Moreover we provide a very efficient algorithm to find this bipartite graph decomposition. Indeed, the algorithm proposed has a complexity comparable to k-core algorithm and we might be able to adapt the optimal algorithm from [Batagelj and Zaveršnik, 2011].

Fast converging re-initialization method for neural networks. The second and final contribution of our thesis from Chapter 6 was, capitalizing on the hypergraph core framework previously described, a re-initialization method for neural networks. More specifically, we computed the weighted hypercore of the graph extracted from the studied neural network after a short period of pretraining and we reinitialize according to the weights of the neural network. This is done by pair of layers, i.e. we are able to apply this framework to every kind of architecture using convolution and linear layers. We provide a framework as generalized as possible but it also outperforms standard initialization and the gradients convergence is faster.

7.2 Future Directions of Research

We discuss in this section the future research interests and directions. We can highlight from every section exciting work that remains to be investigated as well as less related topics that were studied within our research.

First as Chapter 4 has inherited results from Chapter 3 there are many questions that remained unanswered. As pointed out in Chapter 4, there is room for thorough work to prove for the connectivity degeneracy framework the corresponding results found for degree degeneracy. We would also like to investigate vertex connectivity degeneracy as the decompositions produced with this degeneracy are overlapping. Hence very interesting for the major problem that is overlapping clustering.

Moreover our main priority in future work relies on the domain of application tackled in Chapter 6. Indeed using the graph structure of a neural network to acquire information on the learning process is a problem, gathering all the more interest from the machine learning and graph related communities.
Our first exploration direction is a direct consequence of our work, i.e. try to investigate how graph metrics impact learning of a neural network. Precisely, what it means at a neuron level to have high degree, pagerank, core and so on. In fact, seeing how efficiently a naive application of an adapted degeneracy

performed we are very excited to see how neural network gradient variations can be reflected in graph metrics.
The last point worth exploring in the future, is, instead of modeling a single undirected graph at a given moment during the learning, we shall extract a directed graph, showing not only the gradient correction as weights in one direction but also the backpropagation, and eventually studying flows in this graph.

7.3 Epilogue

Research on graphs has proven to be a very exciting domain of research, and many problems have caught up our interest. Working throughout this dissertation was truly a precious experience and attempting to help research move forward brought a rough sense of satisfaction. It also showed that even on known work, we can still always find room for improvement and novel research.

Bibliographie

[Al-garadi et al., 2017] Al-garadi, M. A., Varathan, K. D., and Ravana, S. D. (2017). Identification of influential spreaders in online social networks using interaction weighted k-core decomposition method. *Physica A : Statistical Mechanics and its Applications*, 468 :278–288.

[Alpern and Gal, 2003] Alpern, S. and Gal, S. (2003). *The theory of search games and rendezvous*. International Series in Operations Research & Management Science, 55. Kluwer Academic Publishers, Boston, MA.

[Alspach, 2004] Alspach, B. (2004). Searching and sweeping graphs : a brief survey. *Matematiche (Catania)*, 59(1-2) :5–37 (2006).

[Alvarez-Hamelin et al., 2006] Alvarez-Hamelin, I., Dall'Asta, L., Barrat, A., and Vespignani, A. (2006). Large scale networks fingerprinting and visualization using the k-core decomposition. *Advances in Neural Information Processing Systems*, 18 :41–50.

[Baier et al., 2006] Baier, G., Erlebach, T., Hall, A., Köhler, E., Schilling, H., and Skutella, M. (2006). Length-bounded cuts and flows. In *Automata, languages and programming. Part I*, volume 4051 of *LNCS*, pages 679–690. Springer, Berlin.

[Barrat et al., 2008] Barrat, A., Barthelemy, M., and Vespignani, A. (2008). *Dynamical processes on complex networks*. Cambridge university press.

[Batagelj and Zaversnik, 2003] Batagelj, V. and Zaversnik, M. (2003). An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*.

[Batagelj and Zaveršnik, 2011] Batagelj, V. and Zaveršnik, M. (2011). Fast algorithms for determining (generalized) core groups in social networks. *Advances in Data Analysis and Classification*, 5(2) :129–145.

[Bienstock, 1991] Bienstock, D. (1991). Graph searching, path-width, tree-width and related problems (a survey). In *Reliability of computer and communication networks (New Brunswick, NJ, 1989)*, volume 5 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 33–49. Amer. Math. Soc., Providence, RI.

[Bodlaender et al., 2006] Bodlaender, H. L., Wolle, T., and Koster, A. M. C. A. (2006). Contraction and treewidth lower bounds. *J. Graph Algorithms Appl.*, 10(1) :5–49 (electronic).

[Bollobás, 2013] Bollobás, B. (2013). *Modern graph theory*, volume 184. Springer Science & Business Media.

[Borgwardt and Kriegel, 2005] Borgwardt, K. M. and Kriegel, H. (2005). Shortest-path kernels on graphs. In *Proceedings of the 5th International Conference on Data Mining*, pages 74–81.

[Chen and Schelp, 1993] Chen, G. and Schelp, R. H. (1993). Graphs with linearly bounded ramsey numbers. *Journal of Combinatorial Theory, Series B*, 57(1) :138 – 149.

[Cohen, 2008] Cohen, J. (2008). Trusses : Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*.

[Conte et al., 2004] Conte, D., Foggia, P., Sansone, C., and Vento, M. (2004). Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(03) :265–298.

[Dai et al., 2016] Dai, H., Dai, B., and Song, L. (2016). Discriminative Embeddings of Latent Variable Models for Structured Data. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2702–2711.

[de Vico Fallani et al., 2014] de Vico Fallani, F., Richiardi, J., Chavez, M., and Achard, S. (2014). Graph analysis of functional brain networks : practical

issues in translational neuroscience. *Philosophical Transactions of the Royal Society B : Biological Sciences*, 369(1653) :20130521.

[DeVos et al., 2013] DeVos, M., McDonald, J., Mohar, B., and Scheide, D. (2013). A note on forbidding clique immersions. *Electr. J. Comb.*, 20(3) :P55.

[Diestel et al., 2019] Diestel, R., Hundertmark, F., and Lemanczyk, S. (2019). Profiles of separations : in graphs, matroids, and beyond. *Combinatorica*, 39(1) :37–75.

[Dory, 2018] Dory, M. (2018). Distributed approximation of minimum k-edge-connected spanning subgraphs. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 149–158.

[Dvořák, 2012] Dvořák, Z. (2012). A stronger structure theorem for excluded topological minors. *arXiv :1209.0129*.

[Dvořák, 2013] Dvořák, Z. (2013). Constant-factor approximation of the domination number in sparse graphs. *European Journal of Combinatorics*, 34(5) :833 – 840.

[Erdős and Hajnal, 1966] Erdős, P. and Hajnal, A. (1966). On chromatic number of graphs and set-systems. *Acta Mathematica Hungarica*, 17(1-2) :61–99.

[Exoo, 1983] Exoo, G. (1983). On line disjoint paths of bounded length. *Discrete Mathematics*, 44(3) :317–318.

[Faloutsos et al., 1999] Faloutsos, M., Faloutsos, P., and Faloutsos, C. (1999). On power-law relationships of the internet topology. *ACM SIGCOMM computer communication review*, 29(4) :251–262.

[Fomin and Petrov, 1996] Fomin, F. V. and Petrov, N. N. (1996). Pursuit-evasion and search problems on graphs. In *Proceedings of the Twenty-seventh Southeastern International Conference on Combinatorics, Graph Theory and Computing*, volume 122 of *Congr. Numer.*, pages 47–58.

[Fomin and Thilikos, 2008] Fomin, F. V. and Thilikos, D. M. (2008). An annotated bibliography on guaranteed graph searching. *Theoret. Comput. Sci.*, 399(3) :236–245.

[Gärtner et al., 2003] Gärtner, T., Flach, P., and Wrobel, S. (2003). On Graph Kernels : Hardness Results and Efficient Alternatives. In *Learning Theory and Kernel Machines*, pages 129–143.

[Giannopoulou et al., 2015] Giannopoulou, A. C., Kaminski, M. J., and Thilikos, D. M. (2015). Forbidding kuratowski graphs as immersions. *Journal of Graph Theory*, 78(1) :43–60.

[Giatsidis et al., 2014] Giatsidis, C., Malliaros, F., Thilikos, D., and Vazirgiannis, M. (2014). CORECLUSTER : A Degeneracy Based Graph Clustering Framework. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 44–50.

[Giatsidis et al., 2011] Giatsidis, C., Thilikos, D., and Vazirgiannis, M. (2011). Evaluating cooperation in communities with the k-core structure. In *ASONAM*, pages 87–93.

[Giatsidis et al., 2013] Giatsidis, C., Thilikos, D. M., and Vazirgiannis, M. (2013). D-cores : measuring collaboration of directed graphs based on degeneracy. *Knowledge and information systems*, 35(2) :311–343.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.

[Gomory and Hu, 1961] Gomory, R. E. and Hu, T. C. (1961). Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4) :551–570.

[Grohe et al., 2011] Grohe, M., Kawarabayashi, K.-i., Marx, D., and Wollan, P. (2011). Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 479–488. ACM.

[Grohe et al., 2015] Grohe, M., Kreutzer, S., Rabinovich, R., Siebertz, S., and Stavropoulos, K. (2015). Colouring and covering nowhere dense graphs. In *Graph-Theoretic Concepts in Computer Science - 41st International Work-*

shop, WG 2015, Garching, Germany, June 17-19, 2015, Revised Papers, pages 325–338.

[Haussler, 1999] Haussler, D. (1999). Convolution kernels on discrete structures. *Technical Report*.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers : Surpassing human-level performance on imagenet classification.

[Horváth et al., 2004] Horváth, T., Gärtner, T., and Wrobel, S. (2004). Cyclic Pattern Kernels for Predictive Graph Mining. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 158–167.

[Itai et al., 1982] Itai, A., Perl, Y., and Shiloach, Y. (1982). The complexity of finding maximum disjoint paths with length constraints. *Networks*, 12(3) :277–286.

[Johansson and Dubhashi, 2015] Johansson, F. and Dubhashi, D. (2015). Learning with Similarity Functions on Graphs using Matchings of Geometric Embeddings. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 467–476.

[Johansson et al., 2014] Johansson, F., Jethava, V., Dubhashi, D., and Bhattacharyya, C. (2014). Global graph kernels using geometric embeddings. In *Proceedings of the 31st International Conference on Machine Learning*, pages 694–702.

[Karger and Stein, 1996] Karger, D. R. and Stein, C. (1996). A new approach to the minimum cut problem. *J. ACM*, 43(4) :601–640.

[Kiefer and Wolfowitz, 1952] Kiefer, J. and Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, 23(3) :462–466.

[Kierstead and Trotter, 1991] Kierstead, H. A. and Trotter, W. T. (1991). Planar graph coloring with an uncooperative partner. In *Planar Graphs, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, November 18-21, 1991*, pages 85–94.

[Kirousis and Thilikos, 1996] Kirousis, L. M. and Thilikos, D. M. (1996). The linkage of a graph. *SIAM J. Comput.*, 25(3) :626–647.

[Kondor and Pan, 2016] Kondor, R. and Pan, H. (2016). The Multiscale Laplacian Graph Kernel. In *Advances in Neural Information Processing Systems*, pages 2982–2990.

[Kriege and Mutzel, 2012] Kriege, N. and Mutzel, P. (2012). Subgraph Matching Kernels for Attributed Graphs. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1015–1022.

[Krizhevsky et al., 2009] Krizhevsky, A. et al. (2009). Learning multiple layers of features from tiny images.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553) :436–444.

[LeCun and Cortes, 2010] LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

[Lee et al., 2010] Lee, V., Ruan, N., Jin, R., and Aggarwal, C. (2010). A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336.

[Leskovec et al., 2009] Leskovec, J., Lang, K. J., Dasgupta, A., and Mahoney, M. W. (2009). Community structure in large networks : Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1) :29–123.

[Lick and White, 1970] Lick, D. and White, A. (1970). k-degenerate graphs. *Canadian J. of Mathematics*, 22 :1082–1096.

[Limnios et al., 2020a] Limnios, S., Dasoulas, G., Thilikos, D. M., and Vazirgiannis, M. (2020a). Hcore-init : Neural network initialization based on graph degeneracy. *arXiv preprint arXiv :2004.07636*.

[Limnios et al., 2019] Limnios, S., Paul, C., Perret, J., and Thilikos, D. (2019). Edge degeneracy and admissibility : algorithmic and structural results. *Bordeau Graph Workshop*, pages 204–207.

[Limnios et al., 2020b] Limnios, S., Paul, C., Perret, J., and Thilikos, D. M. (2020b). Edge degeneracy : Algorithmic and structural results. *Theoretical Computer Science*.

[Limnios et al., 2018] Limnios, S., Thilikos, D., and Vazirgiannis, M. (2018). Degeneracy Hierarchy Generator and Efficient Connectivity Degeneracy Algorithm. *not published*.

[Mahé and Vert, 2009] Mahé, P. and Vert, J. (2009). Graph kernels based on tree patterns for molecules. *Machine learning*, 75(1) :3–35.

[Mahjoub and McCormick, 2010] Mahjoub, A. R. and McCormick, S. T. (2010). Max flow and min cut with bounded-length paths : complexity, algorithms, and approximation. *Math. Program.*, 124(1-2) :271–284.

[Matula and Beck, 1983] Matula, D. and Beck, L. (1983). Smallest-last Ordering and Clustering and Graph Coloring Algorithms. *Journal of the ACM*, 30(3) :417–427.

[Matula, 1968] Matula, D. W. (1968). A min–max theorem for graphs with application to graph coloring. *SIAM Reviews*, 10 :481–482.

[Menger, 1927] Menger, K. (1927). über reguläre Baumkurven. *Math. Ann.*, 96(1) :572–582.

[Mishkin and Matas, 2015] Mishkin, D. and Matas, J. (2015). All you need is a good init. *arXiv preprint arXiv :1511.06422*.

[Morris et al., 2019] Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019). Weisfeiler and leman go neural : Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609.

[Nesetril and de Mendez, 2012] Nesetril, J. and de Mendez, P. O. (2012). *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer.

[Neumann et al., 2016] Neumann, M., Garnett, R., Bauckhage, C., and Kersting, K. (2016). Propagation kernels : efficient graph kernels from propagated information. *Machine Learning*, 102(2) :209–245.

[Nešetřil and de Mendez, 2009] Nešetřil, J. and de Mendez, P. O. (2009). Fraternal augmentations, arrangeability and linear ramsey numbers. *European Journal of Combinatorics*, 30(7) :1696 – 1703. EuroComb'07 : Combinatorics, Graph Theory and Applications.

[Nikolentzos et al.,] Nikolentzos, G., Meladianos, P., Limnios, S., and Vazirgiannis, M. A degeneracy framework for graph similarity.

[Nikolentzos et al., 2018] Nikolentzos, G., Meladianos, P., Limnios, S., and Vazirgiannis, M. (2018). A Degeneracy Framework for Graph Similarity. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2595–2601. International Joint Conferences on Artificial Intelligence Organization.

[Nikolentzos et al., 2017a] Nikolentzos, G., Meladianos, P., Rousseau, F., Stavrakas, Y., and Vazirgiannis, M. (2017a). Shortest-Path Graph Kernels for Document Similarity. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1890–1900.

[Nikolentzos et al., 2017b] Nikolentzos, G., Meladianos, P., and Vazirgiannis, M. (2017b). Matching Node Embeddings for Graph Similarity. In *Proceedings of the 31st AAAI Conference in Artificial Intelligence*, pages 2429–2435.

[Nowakowski and Winkler, 1983] Nowakowski, R. and Winkler, P. (1983). Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43(2-3) :235–239.

[Page et al., 1999] Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking : Bringing order to the web. Technical report, Stanford InfoLab.

[Pisanski and Randic, 2000] Pisanski, T. and Randic, M. (2000). Bridges between geometry and graph theory. *MAA NOTES*, pages 174–194.

[Richerby and Thilikos, 2011] Richerby, D. and Thilikos, D. M. (2011). Searching for a visible, lazy fugitive. *SIAM J. Discrete Math.*, 25(2) :497–513.

[Rossi et al., 2015] Rossi, M.-E. G., Malliaros, F. D., and Vazirgiannis, M. (2015). Spread it good, spread it fast : Identification of influential nodes in social networks. In *Proceedings of the 24th International Conference on World Wide Web*, pages 101–102.

[Rousseau and Vazirgiannis, 2013] Rousseau, F. and Vazirgiannis, M. (2013). Graph-of-word and tw-idf : new approach to ad hoc ir. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 59–68.

[Salha et al., 2019a] Salha, G., Hennequin, R., Tran, V. A., and Vazirgiannis, M. (2019a). A degeneracy framework for scalable graph autoencoders. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 3353–3359. AAAI Press.

[Salha et al., 2019b] Salha, G., Limnios, S., Hennequin, R., Tran, V. A., and Vazirgiannis, M. (2019b). Gravity-inspired graph autoencoders for directed link prediction. In *CIKM '19*.

[Schölkopf et al., 2004] Schölkopf, B., Tsuda, K., and Vert, J. (2004). *Kernel Methods in Computational Biology*. MIT press.

[Schrijver, 2003] Schrijver, A. (2003). *Combinatorial optimization : polyhedra and efficiency*, volume 24. Springer Science & Business Media.

[Seidman, 1983a] Seidman, S. B. (1983a). Network structure and minimum degree. *Social networks*, 5(3) :269–287.

[Seidman, 1983b] Seidman, S. B. (1983b). Network structure and minimum degree. *Social Networks*, 5(3) :269–287.

[Shervashidze et al., 2009] Shervashidze, N., Petri, T., Mehlhorn, K., Borgwardt, K. M., and Vishwanathan, S. (2009). Efficient Graphlet Kernels for Large Graph Comparison. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pages 488–495.

[Shervashidze et al., 2011] Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-Lehman Graph Kernels. *The Journal of Machine Learning Research*, 12 :2539–2561.

[Siglidis et al., 2020] Siglidis, G., Nikolentzos, G., Limnios, S., Giatsidis, C., Skianis, K., and Vazirgiannis, M. (2020). Grakel : A graph kernel library in python. *Journal of Machine Learning Research*, 21(54) :1–5.

[Skianis et al., 2020] Skianis, K., Nikolentzos, G., Limnios, S., and Vazirgiannis, M. (2020). Rep the set : Neural networks for learning set representations. In *International Conference on Artificial Intelligence and Statistics*, pages 1410–1420.

[Smola and Schölkopf, 1998] Smola, A. and Schölkopf, B. (1998). *Learning with kernels*. Forschungszentrum Informationstechnik.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout : A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1) :1929–1958.

[Sugiyama and Borgwardt, 2015] Sugiyama, M. and Borgwardt, K. (2015). Halting in random walk kernels. In *Advances in Neural Information Processing Systems*, pages 1639–1647.

[Tang et al., 2008] Tang, J., Zhang, J., Yao, L., Li, J., Zhang, l., and Su, Z. (2008). Arnetminer : Extraction and mining of academic social networks. pages 990–998.

[Vishwanathan et al., 2010] Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. (2010). Graph Kernels. *The Journal of Machine Learning Research*, 11 :1201–1242.

[Wang and Cheng, 2012] Wang, J. and Cheng, J. (2012). Truss decomposition in massive networks. *Proc. VLDB Endow.*, 5(9) :812–823.

[Wang et al., 2017] Wang, L., Long, X., Arends, J. B., and Aarts, R. M. (2017). Eeg analysis of seizure patterns using visibility graphs for detection of generalized seizures. *Journal of neuroscience methods*, 290 :85–94.

[Weißauer, 2019] Weißauer, D. (2019). On the block number of graphs. *SIAM J. Discrete Math.*, 33(1) :346–357.

[Wollan, 2015] Wollan, P. (2015). The structure of graphs not admitting a fixed immersion. *J. Comb. Theory, Ser. B*, 110 :47–66.

[Wuchty and Almaas, 2005] Wuchty, S. and Almaas, E. (2005). Peeling the yeast protein network. *Proteomics*, 5(2) :444–449.

[Yan et al., 2005] Yan, X., Zhou, X. J., and Han, J. (2005). Mining closed relational graphs with connectivity constraints. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 324–333.

[Yanardag and Vishwanathan, 2015] Yanardag, P. and Vishwanathan, S. (2015). Deep Graph Kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374.

[Zhou et al., 2012] Zhou, R., Liu, C., Yu, J. X., Liang, W., Chen, B., and Li, J. (2012). Finding maximal k-edge-connected subgraphs from a large graph. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 480–491.

ECOLE DOCTORALE

Titre : Etude de Dégénérescence de Graph appliqué à l'Apprentissage Automatique Avancé et Résultats Théoriques relatifs

Mots clés : Dégénérescence de Graph, Combinatoire, Théorie des Graphes, Réseaux de Neuronnes, Exploration de Graphe

Résumé : L'extraction de sous-structures significatives a toujours été un élément clé de l'étude des graphes. Dans le cadre d'apprentissages automatiques, supervisés ou non, ainsi que dans l'analyse théorique des graphes, trouver des décompositions spécifiques et des sous-graphes denses est primordiale dans de nombreuses applications comme entre autres la biologie ou les réseaux sociaux.
Dans cette thèse, nous cherchons à étudier la dégénérescence des graphes, en partant d'un point de vue théorique, et en nous appuyant sur nos résultats pour trouver les décompositions les plus adaptées aux tâches à accomplir.C'est pourquoi, dans la première partie de la thèse, nous travaillons sur des résultats structurels des graphes à arête-admissibilité bornée, prouvant que de tels graphes peuvent être reconstruits en agrégeant des graphes à degré d'arête quasi-borné.Nous fournissons également des garanties de complexité de calcul pour les différentes décompositions de la dégénérescence, c'est-à-dire si elles sont NP-complètes ou polynomiales, selon la longueur des chemins sur lesquels la dégénérescence donnée est définie.
Dans la deuxième partie, nous unifions les cadres de dégénérescence et d'admissibilité en fonction du degré et de la connectivité. Dans ces cadres, nous choisissons les plus expressifs, d'une part, et les plus efficaces en termes de calcul d'autre part, à savoir la dégénérescence 1-arête-connectivité pour expérimenter des tâches de dégénérescence standard, telles que la recherche d'influenceurs.
Suite aux résultats précédents qui se sont avérés peu performants, nous revenons à l'utilisation du k-core mais en l'intégrant dans un cadre supervisé, c'est-à-dire les noyaux de graphes. Ainsi, en fournissant un cadre général appelé core-kernel, nous utilisons la décomposition k-core comme étape de prétraitement pour le noyau et appliquons ce dernier sur chaque sous-graphe obtenu par la décomposition pour comparaison. Nous sommes en mesure d'obtenir des performances de l'état de l'art sur la classification des graphes au prix d'une légère augmentation du coût de calcul.
Enfin, nous concevons un nouveau cadre de dégénérescence des degrés s'appliquant simultanément pour les hypergraphes et les graphes biparties, dans la mesure où ces derniers sont les graphes d'incidence des hypergraphes. Cette décomposition est ensuite appliquée directement à des architectures de réseaux de neurones pré-entrainés comme elles induisent des graphes biparties et utilisent le noyau des neurones pour réinitialiser les poids des réseaux de neurones. Ce cadre est non seulement plus performant que les techniques d'initialisation de l'état de l'art, mais il est également applicable à toute paire de couches de convolution et linéaires, et donc applicable à tout type d'architecture.

Title : Graph Degeneracy Studies for Advanced Learning Methods on Graphs and Theoretical Results

Keywords : Graph Degeneracy, Combinatorics, Graph Theory, Neural Networks, Graph Mining

Abstract : Extracting Meaningful substructures from graphs has always been a key part in their study. In machine learning frameworks, supervised or unsupervised, as well as in theoretical graph analysis, finding dense subgraphs and specific decompositions is a primordial task.
In this thesis we aim at studying graph degeneracy, starting from a theoretical point of view, and building upon our results to find the most suited decompositions for the tasks at hand. Hence the first part of the thesis we work on structural results in graphs with bounded edge admissibility, proving that such graphs can be reconstructed by aggregating graphs with almost-bounded-edge-degree. We also provide computational complexity guarentees for the different degeneracy decompositions, i.e. if they are NP-complete or polynomial, depending on the length of the paths on which the given degeneracy is defined.
In the second part we unify the degeneracy and admissibility frameworks based on degree and connectivity. Within those frameworks we pick the most expressive, on the one hand, and computationaly efficient on the other hand, namely the 1-edge-connectivity degeneracy, to experiment on standard degeneracy tasks, such as finding influencial spreaders.
Following the previous results that proved to perform poorly we go back to using the k-core but plugging it in a supervised framework, i.e. graph kernels. Thus providing a general framework named core-kernel, we use the k-core decomposition as a preprocessing step for the kernel and apply the latter on every subgraph obtained by the decomposition for comparison. We are able to achieve state-of-the-art performance on graph classification for a small computational cost trade-off.
Finally we design a novel degree degeneracy framework simultaneously for hypergraphs and on bipartite graphs as they are hypergraphs incidence graphs. This decomposition is then applied directly to pretrained neural network architectures as they induce bipartite graphs and use the coreness of the neurons to re-initialize the neural network weights. This framework not only outperforms state of the art initialization techniques but is also applicable to any pair of layers convolutional and linear thus being applicable however needed to any type of architecture.

Institut Polytechnique de Paris
91120 Palaiseau, France