Higher-order Interactions in Gene Dependency

Ismael Leal

Mansfield College

University of Oxford

A dissertation submitted for the degree of

M.Sc. in Mathematical Modelling and Scientific Computing

Trinity Term 2024

Acknowledgements

This dissertation has been possible thanks to Prof. Alexander Scott. His supervision and guidance during our weekly meetings have played a key role in the completion of this report.

I also thank Novo Nordisk Research Centre Oxford for sponsoring this industry project. They have provided two additional wonderful supervisors, Dr. Lewis Marsh and Dr. M. Lisandra Zepeda-Mendoza. Together with Prof. Scott, the three of them have kept me motivated at all times, even when we seemed to reach a dead end. This dissertation would not have been possible without their help.

Finally, I personally want to thank Dr. Kathryn Gillow for her unconditional support to the whole cohort throughout this whole year. Thank you for making the Mathematical Institute a more homely place.

Abstract

This dissertation applies graph and hypergraph models in the context of gene dependency within cancer cell lines with the aim of exploring hypergraphs as a modelling tool for the biological life sciences. We use novel methods for reconstructing hypergraphs from pairwise data using a Bayesian generative model while integrating pathway-specific subdivisions to manage the computational cost. After the graph and hypergraph construction, several Machine Learning algorithms are trained using structural features from both the graph and hypergraph representations. The performance of these models is then analysed to evaluate the utility of using hypergraphs in comparison to graphs in biological modelling at the molecular level. Additionally, the study integrates CORUM protein complex data to assess the biological significance of the obtained hyperedges. Despite the computational cost and challenges, the results show that hypergraphs have a similar predictive power to that of graphs. However, the computational cost of obtaining a hypergraph is significantly higher than that of obtaining a graph. Hence, the algorithms to reconstruct hypergraphs from data need to be optimised in order to exploit their full potential and make them a useful modelling tool.

Contents

1 Introduction **1**

1 Motivation 1

2 Objectives 2

3 Original contributions 3

4 Structure 3

2 Background and literature review **5**

1 Hypergraph theory 5

2 Gene dependency 10

3 Topological metrics 12

4 Graph and hypergraph embeddings 16

5 Machine learning for node classification 18

3 Methods **22**

1 Graph reconstruction 22

2 Hypergraph reconstruction 28

3 Feature extraction for ML 34

4 ML algorithms 37

4 Results and analysis **41**

1 Graph models 41

2 Hypergraph models 43

3 Pathway-specific essentiality 46

4 Discussion 47

5 Conclusions and future work **50**

1 Conclusion 50

2 Future work 51

A Cell lines of Core CORUM complexes selected for the graph reconstruction **58**

B Computation of betweenness and closeness node centrality for hypergraphs **59**

C Cell lines of Core CORUM complexes found as subsets of hyperedges **61**

1 Introduction

1 Motivation

Graphs are mathematical entities suitable for modelling a variety of systems ranging from biological to social interaction models [1]. They consist of nodes connected by pairwise interactions (edges) and have had a particular success in biological modelling at the molecular level [2]. Yet, molecular interactions often involve groups of molecular entities, rather than pairs [3, 4]. Said interactions are designated higher-order interactions. As a consequence, despite the undoubted success of graphs in the biological sciences in the last decades, a better description of molecular interactions might have more predictive power and unveil new biological insights.

Hypergraphs naturally arise as potential alternatives for modelling higher-order interactions. They are sets of nodes connected by hyperedges: interactions involving an arbitrary number of nodes, thus describing k-way relationships. Hyperedges overcome the limitations that pairwise edges have on capturing group interactions. This framework is not a novel idea [5], but given the effectiveness of graphs it has re-emerged as a promising modelling tool [6, 7]. Unfortunately, most of the mathematical tools and algorithms from graph theory are not directly applicable to hypergraphs [8].

This report is set in the context of modelling gene dependency. Gene dependency is a measure of the extent to which a gene is essential for a certain cell. Genes are considered essential for a cell when their absence impedes the survival or reproduction of the given cell. Acknowledging that cancer is one of the major threats to human lives worldwide [9, 10], improving our understanding of gene dependency and essentiality for cancer cells is a pressing need. In practice, this phenomenon is explored on cell lines, which are collections of cells derived from a single cell. This way, scientists ensure genetic uniformity, avoiding the constant alterations and mutations of tumors [11]. Given the potential descriptive power of hypergraphs, this report aims to assess their usefulness in comparison to graphs in the context of gene dependency for cancer cell lines. For that, we use the latest Cancer Dependency Map (DepMap) release [12]. DepMap is the largest publicly available dataset for cancer cell lines' gene dependency, published by the Broad Institute. An enormous amount of research has been done with it, but to the best of our knowledge it has not been characterised by hypergraphs yet.

In order to assess the performance of both graph and hypergraph representations, we will train several supervised machine learning (ML) models that take structural features from the graph or hypergraph and classify the genes into either essential or non-essential. Firstly, topological metrics from graphs are often used as features for node-classification ML algorithms [13–15]. Therefore, we will feed the ML models with some topological metrics of the nodes from the graph/hypergraph. Additionally, embedding the graph/hypergraph nodes into an n-dimensional continuous space outputs a vector for each node. These vectors, like the topological metrics, form a feature matrix that can be used to train the ML models. The best-performing ML models will be compared, providing insights into graph and hypergraph representations in molecular biology.

2 Objectives

The main goals of this report can be summarised as follows:

- Construct a graph and a hypergraph from the DepMap data.
- Compare graph and hypergraph representations:
  - Compute topological metrics of the nodes from both the graph and the hypergraph.
  - Embed the nodes from both the graph and the hypergraph into a continuous vector space.
  - Train various ML models to predict the essentiality of the genes, using either the topological metrics or the embeddings from each node as features.
  - Compare the performance of both representations.

- Explore the biological significance of the obtained hypergraph by integrating CORUM protein complex data.
- Obtain a list of genes potentially displaying pathway-specific essentiality (see chapter 2, section 2).

3 Original contributions

The graph-reconstruction method used in this thesis is taken from the paper by J. Pan et al. [16]. However, upon exploration of the graph obtained, we modify their definition of functional similarity, as explained in chapter 3, section 1.

This thesis sets off the use of hypergraphs in the context of gene dependency of cancer cell lines, offering a new approach to understanding higher-order gene interactions. The hypergraph reconstruction model used was proposed by J.-G. Young et al. [17], but its computational cost when applied to DepMap data is prohibitively high. Thus, we simplify the data by clustering the genes according to biological processes (pathways). This implies that instead of making essentiality predictions for genes, we make them for gene-pathway pairs, allowing us to make context-dependent predictions for genes. In contrast, published papers tend to focus on the global essentiality of genes.

This report provides a detailed comparison of graph and hypergraph models, offering insight into the strengths and limitations of each for molecular modelling. We find that hypergraph descriptions have a predictive power similar to that of graph descriptions.

We successfully integrate CORUM protein complex data [18] into the hypergraph model, revealing that hyperedges associated with the most reliable ML predictions often include known complexes within their nodes. This finding supports the potential of hypergraphs for identifying biologically significant structures.

Finally, this thesis identifies potential unknown context-dependent essential genes by analysing the most reliable ML predictions. These genes could be essential for some processes, despite being generally classified as non-essential.

4 Structure

Chapter 2 provides the background information needed to set the report into context. In section 1.1, we remind the reader of some definitions from hypergraph theory, analogous to those from graph theory. Given that some algorithms from graph theory are not directly applicable to hypergraphs, in section 1.2 we explore different representations and mappings using hypergraphs, some of which do not conserve the information for higher-order interactions. Section 2 formally introduces the concepts of gene dependency and essentiality, and provides further insight into how the gene dependency data is obtained. In section 3.1 we give definitions for the topological metrics of the nodes that will later be used as ML features. Note these definitions are taken from graph theory. The specific algorithms or definitions that will be applied to hypergraphs are discussed in the next chapter. Section 4 introduces both algorithms used for embedding the graph and the hypergraph, respectively. Finally, section 5 explores the ML models used.

Chapter 3 delves into the methods used in this report. Firstly, sections 1 and 2 discuss the process followed to create a graph and a hypergraph out of the data. Note that the graph and hypergraph reconstruction are considered preparatory steps to achieve the results intended by using ML. Hence, all results regarding these reconstructions will also be presented in this chapter. Section 3 is split into two subsections. The specific topological metrics used in the report and their computation for the case of hypergraphs will be explored in Section 3.1, while section 3.2 analyses the specific embedding algorithm used. Finally, section 4 explains the details of the ML implementation and the selection criteria to obtain the best-performing models.

Chapter 4 will show results from the ML models, both fed with topological metrics and with embeddings. Section 1 selects the best-performing graph-based ML model, and section 2 selects the best hypergraph-based ML model. Section 3 identifies potential context-dependent genes from both the graph- and hypergraph-based models. The interpretation of the results is discussed in section 4.

Finally, chapter 5 synthesises the results and directs possible future work.

2 Background and literature review

1 Hypergraph theory

1.1 Definitions

Before mathematically defining a hypergraph, let us recall the definition for a simple, undirected graph.

Definition 2.1 (Simple, undirected graph)**.** A simple, undirected *graph* $G = (V, E)$ consists of a non-empty set of *nodes* V and a set of *edges* $E \subseteq \{\{x, y\} : x, y \in V, x \neq y\}$. *Weighted* graphs have a non-negative number assigned to every edge, and *unweighted* graphs are such that every weight is 1.

Definition 2.2 (Adjacency matrix)**.** The *adjacency matrix* of an unweighted graph $G = (V, E)$ is given by

$$A_{ij} = \begin{cases} 1, & \{i, j\} \in E, \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Definition 2.3 (Shortest path)**.** A path is a sequence of nodes such that every pair of consecutive nodes is connected by an edge. The shortest path between two nodes in a graph is the path with lowest distance, i.e., with fewest number of edges connecting both nodes.

To define a hypergraph we need to expand the notion of our set of interactions E from a family of pairs to a collection $\{e_i : i \in I\}$ of subsets of V, where each of the subsets e_i contains an arbitrary number of elements from V and I is a finite index set [7]. A *k-interaction* is a set e of nodes such that $|e| = k$, where $|\cdot|$ denotes the cardinality of a set. Hence, edges from graphs are 2-interactions, while a higher-order interaction is any k-interaction such that $k > 2$.

Definition 2.4 (Simple, undirected hypergraph)**.** An undirected *hypergraph* $H = (V, E)$ is defined as a non-empty set of nodes V and a set of hyperedges given by $E = \{e_i \subseteq V : i \in I\}$, where $I = \{1, 2, \dots, |E|\}$. H is a *simple* hypergraph if there are no repeated hyperedges, i.e., $i \neq j \iff e_i \neq e_j$, and no loops: $|e_i| > 1 \, \forall \, i \in I$.

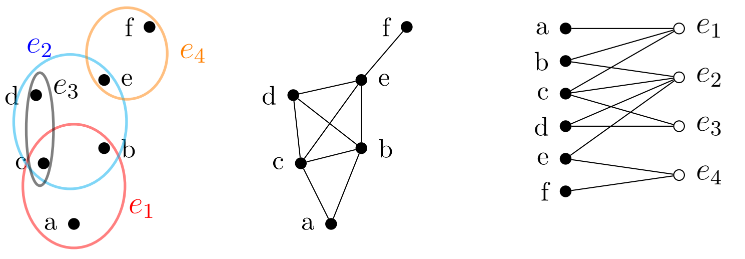
The order and size of hypergraphs are defined analogously to the order and size of graphs. The *order* of a hypergraph $H = (V, E)$ is the cardinality $|V|$ of its node set, and its *size* is the cardinality $|E|$ of its set of hyperedges. The order or *degree* of the node j is defined as the number of hyperedges it belongs to, i.e., $|\{i \in I : j \in e_i\}|$. If the node degree is k for all nodes, the hypergraph is k*-regular.* Furthermore, a hypergraph is k*-uniform* if all its hyperedges have the same cardinality k.

Although listing out the hyperedges gives the complete description of a hypergraph, these higher-order entities introduce several complexities that make it more challenging to compute certain metrics or algorithms compared to traditional graphs. For example, (i) some computations must account for multiple nodes interacting simultaneously, potentially leading to combinatorial explosion and a high computational cost; (ii) some definitions from graph theory have more than one interpretation for hypergraphs, such as the notion of a triangle; (iii) unlike graphs, hypergraphs can show variability in hyperedge size. Several operations and algorithms have been developed for regular and uniform hypergraphs [19, 20], but these special kinds of hypergraphs do not appear often in real-world scenarios. Therefore, we now explore the various representations and expansions that will allow us to compute the topological metrics needed for our hypergraph analysis [8].

1.2 Representations of Higher-Order Interactions

To better understand the importance of the representation choice, let us present a simple example of a system with higher-order interactions. The system will consist of the set of interactions $E = \{e_1 = [a, b, c], e_2 = [b, c, d, e], e_3 = [c, d], e_4 = [e, f]\}$ on the nodeset $V = \{a, b, c, d, e, f\}$.

This system can be described by a vertex-to-hyperedge incidence matrix $Z \in \{0, 1\}^{|V| \times |E|}$, where each row represents a node in V and each column a hyperedge in E. The elements of this incidence matrix are defined as $Z_{ij} = 1$ if $i \in e_j$ for node i, and $Z_{ij} = 0$ otherwise. Equation (2.2) shows the vertex-to-hyperedge incidence



(a) Hypergraph (b) Clique expansion (c) Star expansion

Figure 2.1: Different representations and expansions for the example higher-order system. Note that some information about the hypergraph is lost in (b).

matrix for our example.

$$Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.2}$$

Representing this system as a hypergraph, as shown in Figure 2.1(a), is the most generic and flexible way of describing higher-order systems. Let us now review the different representations and expansions common in the literature.

1.2.1 Clique expansion

Any system with higher-order interactions can be mapped to a graph by expressing each higher-order interaction as a *clique*: a fully-connected set of 2-interactions or normal edges.

Definition 2.5 (Clique expansion)**.** The clique expansion G_{clique} of a hypergraph $H = (V, E)$ is given by $G_{clique} = (V_{clique} = V, E_{clique} = \{\{u, v\} : \exists\, i \in I \text{ such that } \{u, v\} \subseteq e_i\})$.

The graph obtained by this mapping for our example system is shown in Figure 2.1(b). Note that the resulting graph is not necessarily simple: the mapping will result in multiple edges between any two nodes that share more than one hyperedge. A notion of adjacency matrix $A \in \mathbb{Z}^{|V|\times|V|}$, where A_{ij} is the number of hyperedges shared by nodes i and j, can be obtained from the incidence matrix as shown in

Equation (2.3).

$$A = ZZ^T - \text{diag}(\text{diag}(ZZ^T)) = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 2 & 1 & 1 & 0 \\ 1 & 2 & 0 & 2 & 1 & 0 \\ 0 & 1 & 2 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \tag{2.3}$$

where we subtract the diagonal elements to remove self-interactions.

Unfortunately, this representation choice can be insufficient. To see why, just note that the graph representation of our example is indiscernible from another system with interactions $E = \{[a,b], [a,c], [b,c], [b,c], [b,d], [b,e], [c,d], [c,d], [c,e], [d,e], [e,f]\}$. As a consequence, we can not distinguish whether an edge was part of a higher-order interaction or just a pairwise interaction, resulting in information loss.

1.2.2 Star expansion

Another common expansion of hypergraphs, which allows for the application of graph theory, involves the use of *bipartite graphs*.

Definition 2.6 (Bipartite graph)**.** A bipartite graph $BG = (V, E)$ is a simple, undirected graph whose nodes belong to two sets A and B: $V = A \cup B$, where $A \cap B = \emptyset$. Bipartite graphs require that every edge connects a node in A to one in B, so $E \subseteq \{\{a,b\} : a \in A, b \in B\}$.

We can represent a higher-order system $H = (V, E)$ using a bipartite graph where $A = V$ and $B = E$. This representation is also known as the *star expansion* or *incidence graph* [7], given that the nodes representing hyperedges in E are the centres of stars where the outer vertices are the original nodes V that belong to them, as shown in Figure 2.1(c). Note the adjacency matrix $A_* \in \{0,1\}^{(|V|+|E|)\times(|V|+|E|)}$ of the star expansion is given by Equation (2.4) as a block matrix [21].

Definition 2.7 (Star expansion)**.** The star expansion G_* of a hypergraph $H = (V, E)$ is given by the bipartite graph $G_* = (V_* = V \cup E, E_* = \{\{v, e\} : v \in V, e \in E, v \in e\})$.

The star expansion allows the use of well-developed tools from graph theory, as a bipartite graph is, in the end, a graph. For example, the distance between two nodes of the hypergraph can be obtained from this representation, as it is half the distance in the star expansion.

$$A_* = \begin{bmatrix} 0_{|V|} & Z \\ Z^T & 0_{|E|} \end{bmatrix} . \tag{2.4}$$

1.2.3 Motifs

Motifs are recurrent small patterns or subgraphs, expressed as 2-interactions, that may be statistically significant in the network. Note that cliques are a type of motif. Motifs show structural patterns of a network, and can help identifying more information or functionalities than the clique or star expansions, as they recognise interaction patterns in which a node can be involved.

The main disadvantage of motifs is their attainment. In order to identify statistically significant patterns in a system, all the possible motifs need to be explored. This number increases exponentially with the number of nodes, making the use of motifs cumbersome for most real-world applications. Hence, usually the interest in motifs is only restricted to cliques [8].

1.2.4 Simplices

A *simplex*, or hypertetrahedron [22], is the generalisation of the simplest polytope (such as a triangle in 3 dimensions or tetrahedron in 4) to higher dimensions. A k-simplex is a simplex consisting of $k+1$ vertices or nodes.

Definition 2.8 (Simplicial complex)**.** A *simplicial complex* is a set of simplices $\mathcal{K} = \{\sigma_1, \ldots, \sigma_n\}$ such that $\forall\, \sigma \in \mathcal{K}, \forall\, \tau \subseteq \sigma : \tau \neq \emptyset \Rightarrow \tau \in \mathcal{K}$.

Higher-order interactions can be described by simplicial complexes. A simplicial complex $\mathcal{K}$ is a set of simplices such that every non-empty subset of a simplex contained in $\mathcal{K}$ is also in $\mathcal{K}$. Simplicial complexes provide the advantage of not being indiscernible from pairwise descriptions, as is the case for the clique expansion. Equation (2.5) shows the mapping from the hyperedges of a hypergraph e_i to a simplicial complex representation.

$$\mathcal{K} = \bigcup_{i \in I} \{\sigma \subseteq e_i : \sigma \neq \emptyset\} . \tag{2.5}$$

Laplacian operators have been defined for simplicial complexes to describe their structure and topology [23]. Still, their usefulness is computationally restricted by the demand of the existence of all subsets of the simplices within the complex. In order to visualise this restriction, the simplicial complex resulting from our example system is depicted in Figure 2.2. Note that this representation makes the present 4- and 3-interactions explicit, but needs to store all their subsets down to the individual nodes. Thus, the amount of simplices in a simplicial complex scales badly with the system size. There are several examples where the individual or lower-order interactions are a burden rather than useful information. Representing co-authorship as a higher-order interaction shows why: the fact that 3 authors share a paper does not imply the pairwise co-authorships between them. Similarly, if a group of genes share some functionality or interact in a group manner, it does not necessarily mean that the subsets of those genes interact separately.

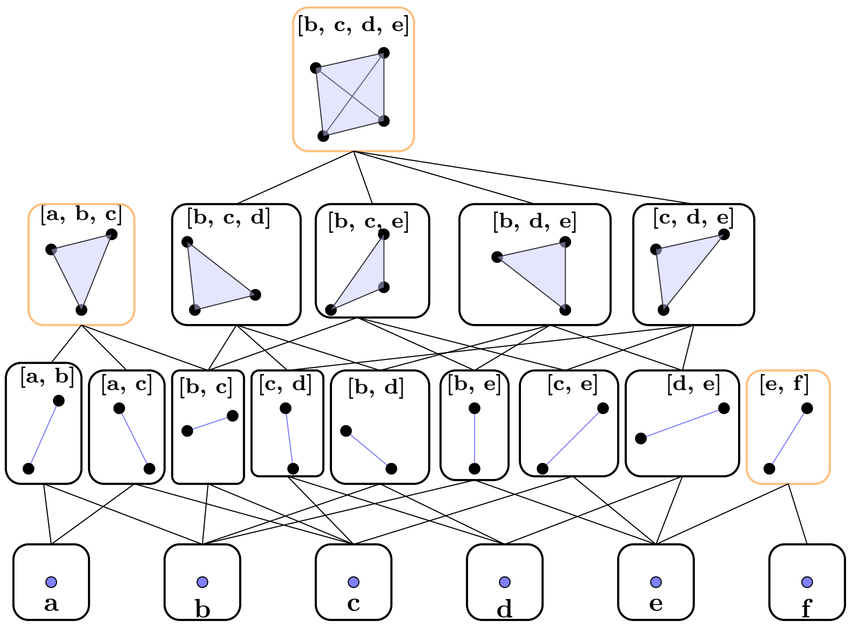


Figure 2.2: Example higher-order system represented as a simplicial complex. Maximal simplices, the highest-dimensional simplices not contained in another simplex, are surrounded in orange.

2 Gene dependency

Genes are DNA segments encoding information for making proteins or molecules with a specific functionality in a cell. These proteins can then for example catalyse biochemical reactions, send signals, or transport substances.

Understanding gene dependency is paramount in the fight against cancer. *Gene dependency* is defined as the degree to which cancer cell lines rely on the expression of specific genes for their survival and proliferation. Specifically, a gene is considered a *dependency* of, or *essential* for, a cancer cell if its inhibition leads to the death of such cell but not of normal, healthy cells. Identifying these dependencies can reveal targets for therapeutic intervention. *Gene co-dependency* describes the fact that some genes are functionally related [24], such that the dependency of a cell or organism on one gene is tightly correlated with its dependency on another gene functionally related to the first one. Gene co-dependency implies that we can mathematically model interactions between genes. Moreover, evidence suggesting that gene co-dependency occurs between groups of genes [25] indicates that hypergraphs can be a promising modelling tool for our purpose.

As will be seen in chapter 3, it is worth introducing the concepts of protein complexes and biological pathways. A *protein complex* is a group of proteins that physically bind together to work as a single molecular unit [26]. The individual proteins conforming it, referred to as subunits, are encoded by genes. Protein complexes play essential roles in all biological processes. A *biological pathway* refers to a series of biochemical reactions that occur within a cell to perform a specific physiological function. A gene may be involved in multiple pathways, performing different functionalities in each. Research suggests that there are multiple levels of essentiality: some genes are essential depending on the context while others are essential across different systems [27]. Thus, the essentiality of some genes might be influenced by the specific pathway in which they are expressed. We refer to this phenomenon as *pathway-specific essentiality.*

CRISPR/Cas9 (Clustered Regularly Interspaced Short Palindromic Repeats), is a gene-editing tool that allows for precise modifications to the genome. Particularly, it cuts DNA strands enabling the inactivation, or *knock-out*, of specific genes. This technology has revolutionized genetic research, allowing scientists to methodically disrupt genes across the genome and study the effects of these knock-outs on cell survival, a process known as high-throughput screenings [28]. The Broad Institute has performed these CRISPR screenings across numerous cancer cell lines [12]. Each knock-out targets and inhibits a specific gene, and their effects determine the amount of dependency of cancer cells on different genes. This large-scale screening effort has provided a detailed map of gene dependencies across different cancer types (Gene Dependency dataset, from now on), offering a resource to further study the way cancer cells work.

The mathematical and computational analysis of CRISPR screening data can reveal complex interactions and dependencies between genes, potentially uncovering genetic vulnerabilities of cancer cell lines. Understanding these dependencies is crucial for developing targeted cancer treatments. By identifying genes essential for cancer cells, scientists can design drugs that specifically inhibit these targets, potentially leading to more effective and safer treatments. The Broad Institute has another published dataset for Inferred Common Essentiality. This dataset includes only those genes that are a dependency across all the cell lines from the Gene Dependency dataset.

In summary, CRISPR screenings and gene dependency analysis are at the front of modern cancer research. The work done by the Broad Institute has provided valuable resources for understanding cancer. By combining the power of CRISPR technology and mathematical analysis, researchers are uncovering the genetic dependencies of cancer cells, opening new pathways for the development of targeted therapies and precision medicine.

3 Topological metrics

Topological metrics are a powerful tool for biological networks, particularly in the oncology field [14]. Over the last decades, the structural or topological features of real-world data have been shown useful to identify relevant elements from the network under consideration [1, 13].

Only small networks are interpretable by humans, with all real-world networks consisting of far too much information for us to process. Consequently, mathematical metrics or measures that can capture features of the complex network data into simple numbers are of interest. These metrics can then be used to train ML models, allowing us to make predictions. Many different measures have been proposed, but the *centrality* measures are some of the most used in biological networks, as they highlight the most relevant or central nodes in a graph. Given that the relevance of a node can be defined in several ways, different centrality measures have been defined over the years [1].

Assuming that topological connectivity implies functional relationship, a notion recently questioned [15], the centrality metrics could be used to classify nodes representing genes as either essential or non-essential using ML models. It has been demonstrated that for life science network analysis, more than one centrality measure has to be considered [29, 30]. Hence, ML models taking various topological measures as feature tables have been used in the last years, and provide us with a way of achieving our goal of inferring gene essentiality.

Here we introduce some centrality measures and some clustering measures only as defined in graph theory. Some of these definitions are not directly transferable to hypergraphs, as can be seen in section 1.2 from this chapter by the importance of the representation or expansion choice. Hence, the implementation of these metrics for hypergraphs is specified in chapter 3, section 3.1.1.

3.1 Degree centrality

The degree of a node is defined in graph theory as simply the number of neighbours it has. More precisely, for a graph $G = (V, E)$ the degree of node v is the cardinality of the open neighbourhood $N(v)$ of node v. This idea can be expressed in mathematical terms as

$$\deg(v) = |\,\{u \in V : \{v, u\} \in E\}\,|. \tag{2.6}$$

The *degree centrality* metric is just a scaling of the node degree. This scaling is such that a node connected to all the other nodes in the graph would have a degree centrality of 1.

Definition 2.9 (Degree centrality)**.** For a simple, undirected graph $G = (V, E)$, the degree centrality of node v is defined as $c_D\,(v) = \dfrac{\deg(v)}{|V| - 1}$.

Degree centrality is sometimes highly informative, albeit being a simple metric. It seems sensible to assume that a gene with a large number of connections in the graph representation of gene co-dependency must be an important gene, having more influence in the whole system than another gene with lower degree centrality.

3.2 Betweenness centrality

The purpose of the betweenness centrality metric is to quantify how frequently a node lies on paths between different pairs of nodes. Consider networks where information packets are sent between pairs of nodes (following shortest paths only). Nodes with a high betweenness score would be those through which many of these packets pass [1]. With this consideration, it seems reasonable that the betweenness centrality measure quantifies the relevance of nodes.

Definition 2.10 (Betweenness centrality)**.** For an undirected graph $G = (V, E)$, the *betweenness centrality* of node v is given by $b(v) = \frac{2}{(|V|-1)(|V|-2)} \sum_{s \neq v \neq t} \dfrac{\sigma_{st}\,(v)}{\sigma_{st}}$, where σ_{st} is the number of shortest paths between the source node s and the target node t, and $\sigma_{st}\,(v)$ counts how many of those paths pass through the node v.

In the general case for an undirected graph where there may be more than one shortest path between two nodes, it is standard to give each path a weight equal to the reciprocal of the number of paths, as shown in Definition 2.10.

3.3 Closeness centrality

Closeness centrality, like betweenness centrality, is defined in terms of shortest paths. Closeness is related to the average distance l_v from a node v to the rest of nodes in a graph. This mean distance is given by

$$l_v = \frac{1}{|V|-1}\sum_{u \neq v} d_{vu}, \tag{2.7}$$

where d_{vu} is the (shortest) distance between nodes v and u. Note we exclude the distance between one node and itself in Equation (2.7). If l_v is small, then node v is, on average, close to other nodes, having then more influence on the rest of the network. Given that the other metrics output large values for relevant nodes, we want the closeness centrality to be related to the reciprocal of l_v. We then reach the definition used for closeness centrality.

Definition 2.11 (Closeness centrality)**.** For an undirected graph $G = (V, E)$, the *closeness centrality* metric of node v is given by $c(v) = \frac{1}{l_v} = \frac{|V|-1}{\sum_{u \neq v} d_{vu}}$, where d_{vu} is the distance between nodes v and u.

3.4 Eigenvector centrality

Some nodes acquire their importance not by being connected to many nodes, but rather by being connected to other nodes that are themselves important in the network. Eigenvector centrality takes this consideration into account. It awards a score proportional to the scores of the neighbours of a node [1].

Defining relevance of a node in terms of other nodes' relevance seems circular. However, a simple mathematical analysis shows that the calculation emerges naturally from the eigenvectors of the adjacency matrix. Let us define this centrality measure x_v for node v to be proportional to the sum of its neighbours' scores, giving

$$x_v = \alpha \sum_{i \in N(v)} x_i, \tag{2.8}$$

where α is the proportionality constant. Note that applying the adjacency matrix automatically selects the neighbours of each node, so that

$$x_v = \alpha \sum_{i=1}^{|V|} A_{vi} x_i. \tag{2.9}$$

Stacking the eigenvector centralities of all nodes into a vector $\mathbf{x} = \left(x_1, \ldots, x_{|V|}\right)^T$ allows us to rewrite Equation (2.9) as a matrix-vector equation

$$\mathbf{Ax} = \alpha^{-1}\mathbf{x}. \tag{2.10}$$

By setting $\alpha^{-1} := \lambda$, we have arrived at an eigenvector-eigenvalue equation, where the centrality scores we defined for each node are now the entries of some eigenvector $\mathbf{x}$. To select one of the $|V|$ eigenvectors of the adjacency matrix, the Perron-Frobenius theorem [31, 32] is of utmost help. It states that a real square matrix with positive entries (such as the adjacency matrix) has one eigenvector with its entries strictly positive, and that eigenvector is the one associated to the dominant eigenvalue. It seems reasonable to select said eigenvector so that all our eigenvector centrality scores are non-negative [1].

Definition 2.12 (Eigenvector centrality)**.** For an undirected graph, the *eigenvector centrality* metric for node v is defined as the v-th entry of the normalised dominant eigenvector of the adjacency matrix.

Fortunately, the power method provides a simple algorithm to precisely approximate this vector with a low computational cost [33]. Note that for directed graphs, the adjacency matrix loses its symmetry, making the eigenvector centrality calculation more nuanced.

3.5 Clustering coefficient

The local clustering coefficient is a measure of the level of union in the neighbourhood of a node. For some node v, it is the fraction of triangles containing v over the total number of possible triangles given its neighbours. In other words, it measures the fraction of loops of length three formed by each node.

Definition 2.13 (Local clustering coefficient)**.** For an undirected graph $G = (V, E)$, the *local clustering coefficient* of node v with a set of neighbours $N(v)$ is given by

$$\frac{2|\left\{e_{ij} : i, j \in N(v), e_{ij} \in E\right\}|}{k_v \left(k_v - 1\right)}, \tag{2.11}$$

where $k_v = |N(v)|$ and e_{ij} is an edge from E connecting nodes i and j.

3.6 Bottleneck coefficient

In order to define the bottleneck coefficient, we first introduce the notion of a shortest path tree. A shortest path tree T_s rooted at node s in a fully-connected graph $G = (V, E)$ is a spanning tree in which, for every node $v \in V$, the path from s to v is the shortest path from G.

The purpose of the bottleneck coefficient is to measure how often a node v acts as a bottleneck across all shortest path trees, i.e., how often it is a critical point through which a significant portion of the network's information must pass. A high bottleneck value for a node indicates that it frequently appears in many shortest path trees rooted at different nodes. We take its definition from [34].

Definition 2.14 (Bottleneck coefficient)**.** For an undirected graph $G = (V, E)$, the *bottleneck coefficient* of a node v is given by $\mathrm{bn}(v) = \sum_{s \in V} p_s(v)$, where $p_s(v) = 1$ if more than $|V|/4$ paths from the shortest path tree T_s rooted at node s pass through v; otherwise, $p_s(v) = 0$.

Both the bottleneck coefficient and the betweenness centrality measure the role of each node in facilitating information flow. The main difference is that the betweenness centrality considers shortest paths between every pair of nodes in a graph, while the bottleneck coefficient considers the shortest path tree rooted at every node, and counts nodes meeting some threshold. This means that the betweenness centrality provides a global perspective of node relevance, in contrast to the more localised measure given by the bottleneck coefficient.

4 Graph and hypergraph embeddings

4.1 Node2Vec algorithm

In this section we present an overview of *node2vec*, an embedding algorithm for graphs [35]. Embedding algorithms, used to make predictions over the nodes and edges of a graph, learn continuous feature representations of the nodes in a graph. In graph modelling, the structure of graphs contains information that can be harnessed for several ML tasks. The traditional feature engineering techniques struggle with capturing the complexity of the interactions within these networks. Usually, they lack flexibility when exploring types of node neighbourhoods. *node2vec* addresses this limitation by maximising the likelihood of preserving graph neighbourhoods of nodes.

The goal of embedding a graph is to map the nodes to a continuous vector space where nodes with similar roles or structural characteristics are close to each other. The algorithm does this mapping by simulating biased random walks over the graph and later applying the Skip-gram model to said walks [36].

Definition 2.15 (Biased random walk)**.** Consider an unweighted graph $G = (V, E)$. Let π_{vx} be the unnormalised transition probability associated with the edge $(v, x) \in E$, and c_i denote the i-th node [35]. A *biased random walk* on the graph G is a Markov process [37] in which the probability of transitioning from node v to node x at step i is given by:

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E, \\ 0 & \text{otherwise,} \end{cases} \tag{2.12}$$

where $Z = Z\,(v)$ is a normalisation constant. The term π_{vx} introduces the bias, favouring certain transitions over others.

The transition probability π_{vx} is defined in the *node2vec* algorithm in terms of two parameters, p and q, which control the likelihood of returning to an already-visited node or moving onto unvisited parts of the graph. This biased random walk is a second-order Markov process, given that the probability at each step depends on the current node and the previous one. For a biased random walk through an unweighted graph that has transitioned from node t to node v and is now deciding its next step, the transition probability [35] is defined as

$$\pi_{vx} = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0, \\ 1 & \text{if } d_{tx} = 1, \\ \frac{1}{q} & \text{if } d_{tx} = 2, \end{cases} \tag{2.13}$$

where d_{tx} is the distance or number of edges between nodes t and x. From Equation (2.13), a high value of the parameter p reduces the likelihood of passing through already-visited nodes, while a low value would result in a random walk that keeps close to the initial node. On the other hand, a high q value biases the walk to nodes close to t, whereas a low q value encourages exploring more distant nodes, leading to a broader neighbourhood exploration. *node2vec* tunes this parameters and allows the algorithm to capture both local and global structural relationships within the graph.

After the random walks the Skip-gram model is applied. It is tasked with predicting the neighbourhood for an input node from the random walk. Over many iterations, the resulting embeddings place those nodes that frequently appear together in random walks closer together in the vector space. These embeddings are the ones that can later be used for our node classification task.

4.2 HNHN embeddings

We now provide an overview of the *Hypergraph Networks with Hyperedge Neurons (HNHN)* model [38]. *HNHN* is a Neural Network (NN) method for learning representations from hypergraphs. Unlike traditional graph neural networks that rely on adjacency matrices [39], *HNHN* uses the hypergraph incidence matrix (as defined in chapter 2, section 1.2) and non-linear activation functions with a novel normalisation approach. In this model, both nodes and hyperedges are treated as fundamental entities, each with their own embeddings, X_V and X_E, respectively.

The main innovation in *HNHN* is the alternating update of the node and hyperedge embeddings through linear and non-linear transformations. The embeddings need to be initialised and then iteratively refined following Equations (2.14) and (2.15). Thus, the forward pass of this model alternates between updating the node and the hyperedge embeddings.

$$X'_E = f\left(Z^T X_V W_E + b_E\right), \tag{2.14}$$

$$X'_V = f\left(Z X'_E W_V + b_V\right), \tag{2.15}$$

where W_E and W_V are weight matrices, b_E and b_V are biases, and $f(\cdot)$ is a non-linear activation function, typically ReLU. The final output is a set of node embeddings X'_V.

The embeddings are scaled by the degrees of the nodes and hyperedges, to ensure that their influence is balanced across the hypergraph, preventing the embeddings from being dominated by entities with high degree [38].

5 Machine learning for node classification

Node classification is a key problem in graph analysis whose goal is to predict the label of a node based on its attributes and connections. ML is a tool suited for this task. Also, every ML model has a *confidence score* measuring the probability that a given input has been correctly classified. Thus, predictions with a higher confidence score give more insight into those specific inputs. In this section, we will introduce several popular supervised ML models used for node classification. However, we first provide the definitions for several ML performance metrics often used in these contexts.

Definition 2.16 (Precision)**.** *Precision* is a metric that quantifies the fraction of instances identified as positive by the model that are actually positive. It is given by

$$\text{Precision} = \frac{\text{True positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}. \tag{2.16}$$

Definition 2.17 (Recall)**.** *Recall*, or *sensitivity*, measures the proportion of true positive instances that are correctly identified by the model. It can be computed as

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}. \tag{2.17}$$

Definition 2.18 (F1-Score)**.** The *F1-score* is defined as the harmonic mean of the precision and recall metrics, accounting for both false positives and false negatives. Its expression is

$$\text{F1-Score} = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{2.18}$$

5.1 Logistic Regression (LR)

Logistic Regression is one of the simplest algorithms, often used for binary classification (although it can be extended to multi-class classification). For binary classification, LR models the probability that a node v belongs to a particular class $y_v \in \{0, 1\}$ based on the input vector of features $\mathbf{x}_v$. It maps the log-odds to a probability using the sigmoid function $\sigma(\cdot)$,

$$P(y_v = 1 \mid \mathbf{x}_v) = \sigma\left(\mathbf{w}^T \mathbf{x}_v\right) = \frac{1}{1 + \exp\left(-\mathbf{w}^T \mathbf{x}_v\right)}, \tag{2.19}$$

where $\mathbf{w}$ is the vector of weights. This leads to the minimisation problem shown in Equation (2.20).

$$\min_{\mathbf{w}} \left[-\sum_{v=1}^{n} \left[y_i \log\left(\sigma\left(\mathbf{w}^T \mathbf{x}_v\right)\right) + (1 - y_i) \log\left(1 - \sigma\left(\mathbf{w}^T \mathbf{x}_v\right)\right) \right] + \frac{\lambda}{2} \|\mathbf{w}\| \right], \tag{2.20}$$

where λ is a regularisation parameter that penalises overfitting.

5.2 Random Forest (RF)

Random Forest is an *ensemble learning* algorithm. Ensemble learning techniques are ML models that aggregate several learners with the aim of producing better predictions. RF builds multiple decision trees during its training and aggregates their predictions.

Decision trees, or *classification trees* for discrete outputs, are supervised ML models that operate by recursively partitioning the feature space into increasingly homogeneous subsets, making decisions based on the input values. The decision process can be visualised as a tree with internal nodes, external nodes (leaves), and branches. The internal nodes represent tests on a feature, while the leaves represent an output label. Finally, each branch represents the outcome of a test.

In a RF model, each tree has the entire dataset as root node. The algorithm selects a feature j and a threshold t to split the data into two subsets according to the value of that feature, as in Equation (2.21). We denote by $\mathbf{x}^{(i)}$ the vector of features for the i-th input, and by $x_j^{(i)}$ the j-th feature from the i-th input. The best split is determined by impurity measures such as entropy.

$$\mathcal{D}_{left} = \left\{ \left(\mathbf{x}^{(i)}, y_i\right) : x_j^{(i)} \leq t \right\}, \qquad \mathcal{D}_{right} = \left\{ \left(\mathbf{x}^{(i)}, y_i\right) : x_j^{(i)} > t \right\}. \tag{2.21}$$

The process described above is repeated recursively for each internal node until a stopping criterion is met. Examples of stopping criteria are reaching a maximum depth or having a minimum number of samples in each leaf. Once the tree is grown, each leaf node is assigned the class with more instances it contains [40]. In order to make a prediction, each input gets classified to a certain leaf, and hence to some output class, according to its features and the best splits decided during training.

Combining multiple decision trees improves the predictive power of the model. The main idea is to reduce the variance of the model by averaging predictions of several decision trees. A more robust model is obtained if each of the decision trees are trained on different subsets of the data [41]. In order to introduce further diversity among the trees, at every split of each tree a random subset of features is selected. Hence, each best split is determined only within the subset of features under consideration. Once all trees are grown, RF classifies an instance $\mathbf{x}_v$ by majority voting. If the task is regression instead of classification, the final prediction averages the predictions of each of the individual tree predictions.

5.3 Neural Networks (NNs)

Neural networks can capture complex, non-linear relationships in the data. A feed-forward NN consists of layers of neurons, where each neuron computes a weighted sum of its inputs and then applies a non-linear activation function to the weighted sum before passing the result to the next layer. The output of the l-th layer $\mathbf{h}^{(l)}$ is computed in terms of the output from the previous layer $\mathbf{h}^{(l-1)}$, the weight matrix $\mathbf{W}^{(l)}$ and the bias vector $\mathbf{b}^{(l)}$ of the current layer, and the activation function $f(\cdot)$ as shown in Equation (2.22).

$$\mathbf{h}^{(l)} = f\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right). \tag{2.22}$$

Despite the capacity of neural networks to learn relevant features from complex data, they need large amounts of training data, which is usually limited for biological tasks.

5.4 Support Vector Classifier (SVC)

The Support Vector Classifier is a learning model that tries to find the hyperplane that best separates the classes in the feature space. For linearly separable data, the optimal hyperplane is given by $\mathbf{w}^T\mathbf{x} + b = 0$, where $\mathbf{w}$ is the normal vector to the hyperplane, and b is a bias term. The aim is to maximise the distance between the hyperplane and the closest data points, or support vectors. For non-linearly separable data, SVC applies a different kernel function to implicitly map the data into a higher-dimensional space where a linear separation is viable.

5.5 K-Nearest Neighbours (KNN)

K-Nearest Neighbours is an algorithm that assigns a node v to the class most common among its k nearest neighbours in the feature space. Usually, how close two nodes are is measured with a distance metric such as the Euclidean distance. Given a set of labelled nodes, KNN predicts the label $\hat{y}_v$ of node v as given by Equation (2.23).

$$\hat{y}_v = \text{mode}\left\{y_j : j \in N_k(v)\right\}, \tag{2.23}$$

where $N_k(v)$ is the set of k nearest neighbours of node v according to the corresponding distance metric.

3 Methods

In this chapter, we present the steps followed to achieve the results presented in chapter 4.

1 Graph reconstruction

In this section, we aim to build an unweighted graph representing gene co-dependency from the CRIPSR Gene Dependency dataset.

The data consists of gene dependency scores in the interval $[0, 1]$, where each value represents the score of knocking out a gene from a cancer cell line. Columns in the dataset represent different genes, and rows represent different cancer cell lines. Values close to 1 indicate that the proliferation of cancer cells in the cell line is largely dependent on the gene, showing the knock-out causes a significant decrease in cellular fitness. Values close to 0 indicate almost no relationship between the cell line's fitness and the expression of the gene.

In order to build an unweighted graph from gene dependency data, it has been proposed [16] to represent genes as nodes where two genes share an edge if some functional similarity measure falls above a certain threshold.

1.1 Functional similarity

J. Pan et al. [16] define functional similarity between two genes using their gene dependency scores across all cell lines, also called *fitness profiles.* They take the Pearson correlations of the fitness profiles for all pairwise combinations of genes, rank-order normalise all the correlations obtained for each gene, and symmetrise the resultant matrix of ranks [16]. Then, by selecting a threshold, all those pairs of genes with a functional similarity value above said threshold are considered to share an edge. Note we normalise the ranks so that their range is always in $[0, 1]$.

To get an intuition of why using the correlations can be a good idea, let us compare the fitness profiles of different genes. This definition of functional similarity assumes that genes with similar functions have similar patterns of dependency across different cell lines. This similarity would be reflected in their fitness profiles, resulting in highly-correlated profiles. Figure 3.1 compares randomly-chosen genes, while Figure 3.2 compares genes belonging to a protein complex present in eukaryotic mitochondria acting on human heart tissue (F1F0-ATPase complex) [18]. Hierarchical clustering with complete linkage was used to generate the dendrograms on both figures. These dendrograms show the distance between pairs of genes in the x-axis, where we have defined the distance d_{ij} between two genes i and j in terms of the absolute value of the pairwise correlation r_{ij} between them, as $d_{ij} = 1 - |r_{ij}|$.

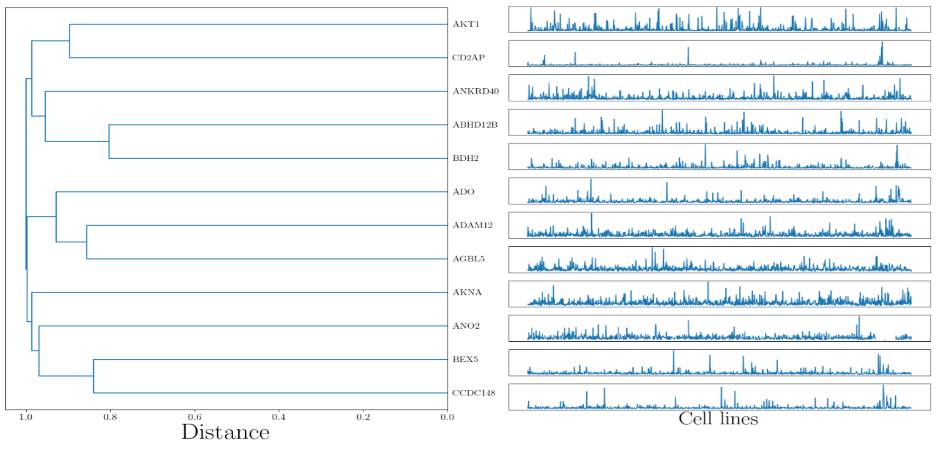


Figure 3.1: Dendrogram (left) and fitness profiles (right) for 12 randomly-chosen genes from within the Gene Dependency dataset. Note the large distances (left) and the dissimilar profiles (right).

Comparing both Figures 3.1 and 3.2, we deduce that correlation along cell lines can indeed be used as a functional similarity measure. Both the dendrogram (left) and the gene profiles (right) in Figure 3.2 show a significant correlation among the genes; this correlation is absent in Figure 3.1, as expected. Hence, we construct a graph by creating an edge between any two genes that have the rank of the absolute value of their correlation coefficient above some threshold.

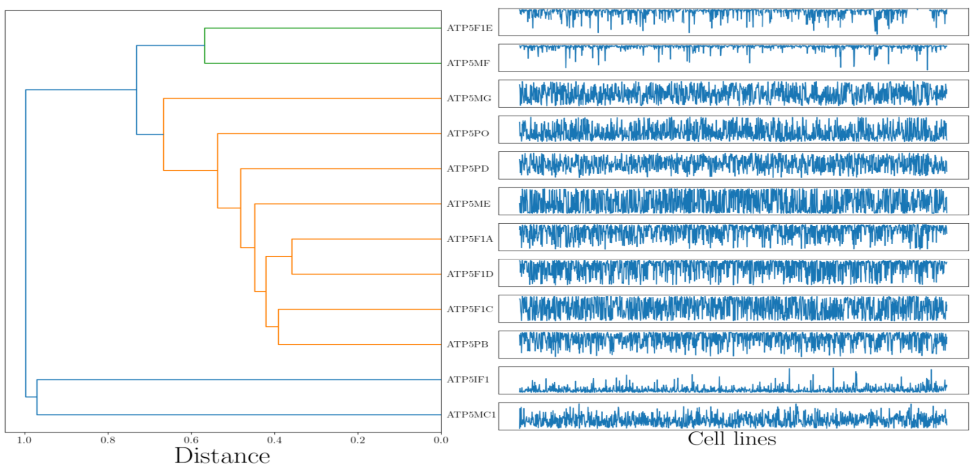


Figure 3.2: Dendrogram (left) and fitness profiles (right) for 12 genes belonging to the F1F0-ATPase, mitochondrial complex. Note the small distances (left) and the similar profiles (right).

1.1.1 Graph exploration

By following the procedure described above, different graphs are obtained for different thresholds. Figure 3.3 shows how the graph density, defined as $d = |E|/(^{|V|}C_2) = 2|E|/\left(|V|(|V|-1)\right)$ varies for 50 different thresholds.

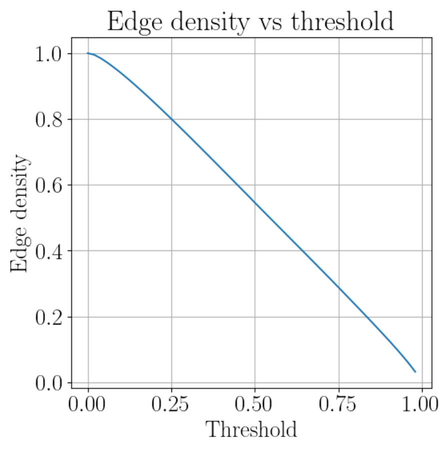


Figure 3.3: Edge density as a function of the threshold, using the ranked correlations.

Given that the graph obtained from the CRISPR Gene Dependency dataset has $|V| = 18443$ nodes, Figure 3.3 implies that the graph is dense for large thresholds. For a threshold of 0.8, 1 out of every 5 possible connections between all genes exists.

This occurs due to the ranking of the absolute-valued correlations. After taking their ranks, the number of existing edges varies linearly with the threshold. This linearity is expected, given that ranking imposes a uniform distribution over the threshold values.

Here, we propose altering the definition given by J. Pan et al. [16] of functional similarity, to only the absolute values of the pairwise Pearson correlations between genes. In other words, we will not rank-order normalise the absolute correlations. The results are shown in Figure 3.4. In order to give the reader a visual intuition of the graph size, Figure 3.4 (right) also shows the edge number as a function of the threshold.

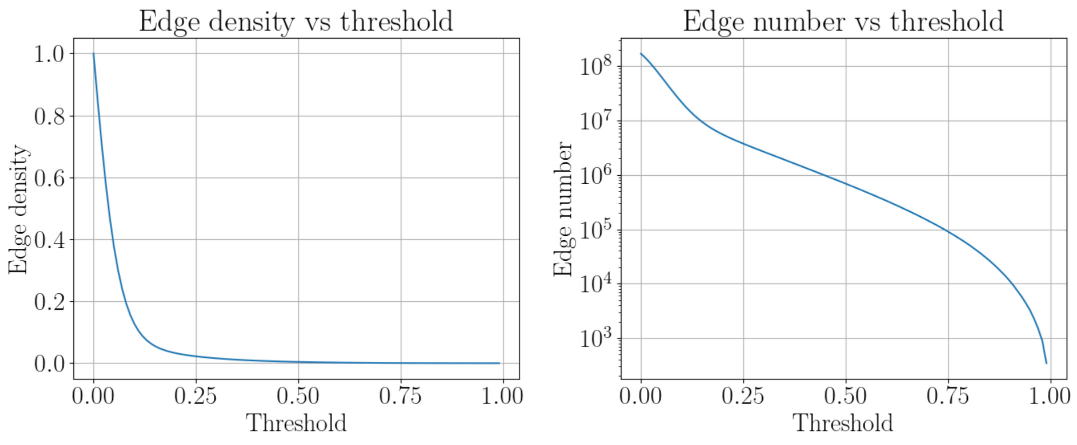


Figure 3.4: Edge density vs threshold (left) and edge number vs threshold (right), without ranking the correlations.

Using the absolute values of the correlations directly results in a non-linear, more natural shape. Figure 3.4 (left) shows a steep decline in edge density at low threshold values, followed by a more gradual slope after a certain point, between 0.15 and 0.2. The steep decline suggests a null distribution of edges. Here, slight perturbations of the threshold lead to significant changes in edge density, indicating that many weak relations, likely noise, are being filtered in or out. The slope of the edge distribution decreases notably after 0.2. We postulate that most edges included at this threshold correspond to true co-dependencies, with a small number of false positives being included.

1.1.2 Threshold selection

In order to assess whether the graphs obtained with the proposed method are accurate with the state-of-the-art research, we perform an analysis to find known Core CORUM protein complexes [18] as statistically-significant dense node groups. This process also allows us to choose a threshold.

The process focuses on evaluating the internal to external *edge density ratio* (EDR) of node groups representing protein complexes within a graph. For each protein complex, we calculate its EDR and compare this value to a null distribution generated by a configuration model. This model rewires the graph multiple times while preserving the degree sequence, creating a distribution of EDRs for each complex. Then we obtain a p-value for each complex, representing the fraction of EDR values from the null distribution that exceed the original value. This value is adjusted to avoid p-values of zero by computing $(t+1)/(n+1)$, where t is the number of instances from the null distribution that exceed the observed EDR, and n is the total number of instances in the null distribution. Finally, we apply a False Discovery Rate (FDR) correction and consider those complexes with a p-value below the 0.05 threshold as significant. The reader interested in further details about the FDR correction may consult Y. Benjamini [42, 43].

J. Pan et al. [16] use the internal to external edge density ratio (EDR) as in Definition 3.1. The *internal edge density* (IED) of a protein complex is the number of internal edges (existing edges connecting two nodes belonging to the complex) divided by the total number of possible internal edges. They define an external edge as an edge connecting a node from the complex to a node outside the complex. Hence, the *external edge density* (EED) of a complex is the ratio of existing external edges to total number of possible external edges. Finally, the EDR is the ratio of the IED to the EED.

Definition 3.1 (Internal to external EDR)**.** Let E be the edge set of an unweighted graph $G=(V,E)$. Consider a set $\mathcal{N}$ of nodes representing a protein complex. The number of edges internal to the complex is $n_{\text{int}} = |\{\{u,v\} : u,v \in \mathcal{N}, \{u,v\} \in E\}|$. Similarly, the number of edges connecting a node within the complex to a node outside it is $n_{\text{ext}} = |\{\{u,v\} : u \in \mathcal{N}, v \notin \mathcal{N}, \{u,v\} \in E\}|$. The internal to external EDR is then given by

$$\text{EDR} = \frac{2n_{\text{int}}/\left(\left(|\mathcal{N}|\right)\left(|\mathcal{N}|-1\right)\right)}{n_{\text{ext}}/\left(|\mathcal{N}|\left(|V|-|\mathcal{N}|\right)\right)}. \tag{3.1}$$

Algorithm 1 Single graph rewiring and EDR obtention using *graph-tool* library.

```
Require: graph-tool Graph object for some threshold value, list of gene names for
    the CORUM complex of interest, list of internal nodes internal_nodes, list of
    external nodes external_nodes.
Ensure: all gene names from CORUM complex appear on CRISPR dataset.
 1: Initialise internal and external edge counts $n_{\text{int}} \leftarrow 0, n_{\text{ext}} \leftarrow 0$.
 2: Create degrees_list, holding the degrees of the internal nodes. Additionally, its
    last element is the sum of degrees of all the external nodes.
 3: Store number of internal and external nodes, $|V_{\text{int}}|$ and $|V_{\text{ext}}|$, respectively.
 4: Compute the number of possible internal edges: $n_{\text{total,int}} = |V_{\text{int}}| * (|V_{\text{int}}| - 1)/2$.
 5: Compute the number of possible external edges: $n_{\text{total,ext}} = |V_{\text{int}}| * |V_{\text{ext}}|$.
 6: for node in internal_nodes do
 7:     $deg \leftarrow$ degree of current node from degrees_list.
 8:     while $deg > 0$ do
 9:         Compute probabilities that a stub from this node will connect to each of
    the other internal nodes, and to any of the external nodes (using the sum of
    external degrees).
10:         Randomly choose a target node $t$ based on probabilities.
11:         $deg_{\text{t}} \leftarrow$ degree of target node (or of external set of nodes) from degrees_list.
12:         if $t$ in internal_nodes then
13:             $n_{\text{int}} \leftarrow n_{\text{int}} + 1$.
14:         else
15:             $n_{\text{ext}} \leftarrow n_{\text{ext}} + 1$.
16:         end if
17:         $deg \leftarrow deg - 1$ and $deg_{\text{t}} \leftarrow deg_{\text{t}} - 1$.
18:     end while
19: end for
20: IED = $n_{\text{int}}/n_{\text{total,int}}$ and EED = $n_{\text{ext}}/n_{\text{total,ext}}$.
21: EDR = IED / EED
```

Unfortunately, given the size of the graph, using built-in functions from graph-specific libraries to apply the configuration model 10,000 times takes an enormous amount of time and computational resources. We observed that to find the p-value of a certain complex, the specific rewirings obtained for the edges outside the node group of interest are not needed. Hence, we propose Algorithm 1 to perform the same process without the need of rewiring the whole graph. We treat all the nodes external to the complex as one single node with a degree equal to the sum of all the degrees of external nodes.

Algorithm 1 allows the use of the configuration model without a whole graph rewiring. Applying this algorithm 10,000 times for each of the protein complexes of interest (see Table A.1), we obtain a list of p-values. After applying an FDR correction, those complexes with a p-value above the 0.05 cutoff are considered to have a statistically-significant EDR. We consider such complexes as present in our graph. Figure 3.5 shows the fraction of present Core CORUM complexes using this method.

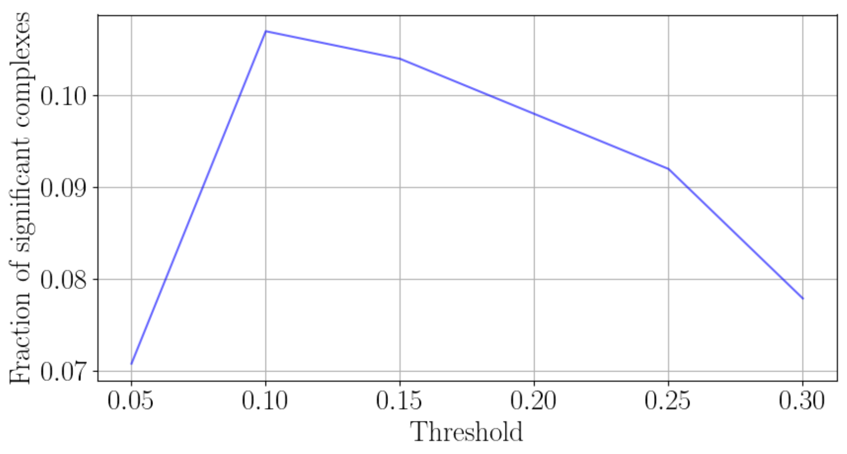


Figure 3.5: Fraction of statistically significant complexes found against the absolute correlation threshold.

The largest number of found complexes occurs for a threshold of 0.1 or 0.15. Given that the turning point from Figure 3.4 described earlier occurs around 0.2, we will select a threshold of 0.2.

2 Hypergraph reconstruction

In this section we outline the steps followed to describe the human genes from the CRISPR Gene Dependency dataset in terms of higher-order interactions. The method reconstructs a hypergraph from a graph. We will use the graph obtained in section 1 as an input in this section.

2.1 Method to infer higher-order interactions

We now briefly describe the method to recover higher-order interactions from pairwise network data, developed by Young, Petri, and Peixoto in [17] and implemented in the *graph-tool* Python library [44]. This approach is particularly apt for identifying group interactions in complex systems, such as the set of human genes. The method employs a Bayesian generative framework to infer latent higher-order interactions, i.e., hyperedges that are not explicitly recorded in the network data. Furthermore, it is based on the principle of parsimony, which states that the most acceptable explanation of a phenomenon is the simplest.

At the core of the generative model is the projection component $P(G|H)$ that describes how network data G is generated when underlying higher-order interactions H exist. This component is built based on the assumption that nodes are connected in G if and only if they share at least one hyperedge in H. The prior $P(H)$ is obtained using the Poisson Random Hypergraphs Model (PRHM). We refer the interested reader to Darling and Norris [45]. With these two quantities, the posterior probability $P(H|G)$ can be computed. In order to sample from this distribution, Young et al. use a Metropolis-Hastings Markov Chain Monte Carlo (MCMC) algorithm. This way, they achieve a random walk over the space of possible hypergraphs given by the posterior distribution. As noted in [17], different initialisations of the MCMC algorithm will probably deduce different hyperedges. Here we follow their recommendation to use maximal clique initialisation, that is, setting one hyperedge for each maximal clique.

The main focus of this method, maximising parsimony, is achieved by maximising the posterior probability of the hypergraph, i.e., by describing the data in the most compact representation of hyperedges. This provides a rigorous criterion for choosing the most likely hypergraph: sample from $P(H|G)$ with the MCMC algorithm and select the hypergraph with the highest posterior probability, as defined in [17].

2.2 Pathway subdivision

In [17], Young et al. state that despite enumerating maximal cliques being an NP-hard problem, enumeration was not a problem in their experiments. Perhaps by cause of the order ($|V| = 18,443$) and size ($|E| \approx 5.5 \cdot 10^6$) of our input graph G at the 0.2 threshold, enumerating maximal cliques is indeed too computationally expensive in our case. As a solution to this computational limitation, we resolved to split the input graph G into different subgraphs according to gene pathways. This partitioning results in attainable maximal clique enumerations to get a hypergraph from each subgraph.

This approach relies in the inherent modularity of biological systems. Biological evidence supports that genes are usually organised into functional pathways [46, 47]. Therefore, we can expect the most relevant higher-order interactions to occur among genes from same pathways, making this a reasonable and computationally feasible subdivision. The current scientific consensus on human pathways and the genes they include can be downloaded from the Reactome database [48]. The “UniProt to pathways” Reactome database lists, for each gene, the different pathways it is involved in. After some preprocessing, we mapped each pathway to all the genes involved with it. We filter the dataset for only those entries that have been reviewed and then remove the pathways represented by subgraphs without any edges. Given that our focus is on the higher-order interactions, those pathways/subgraphs consisting of one or two genes were grouped together into an “Unmapped” group. The genes with a NaN pathway were also added to the Unmapped subgraph. Lastly, there are 11 genes from the CRISPR dataset that did not feature a reviewed entry in the Reactome dataset which were grouped into an “Unknown” subgraph. This process results in 1,233 subgraphs. Figure 3.6 shows the distribution of the subgraph orders.

Note that some genes in the Reactome database are part of several pathways. Thus, the ML models will be fed a set of topological metrics or an embedding for each gene-pathway pair, instead of for each gene. This consideration will be further discussed in section 4.

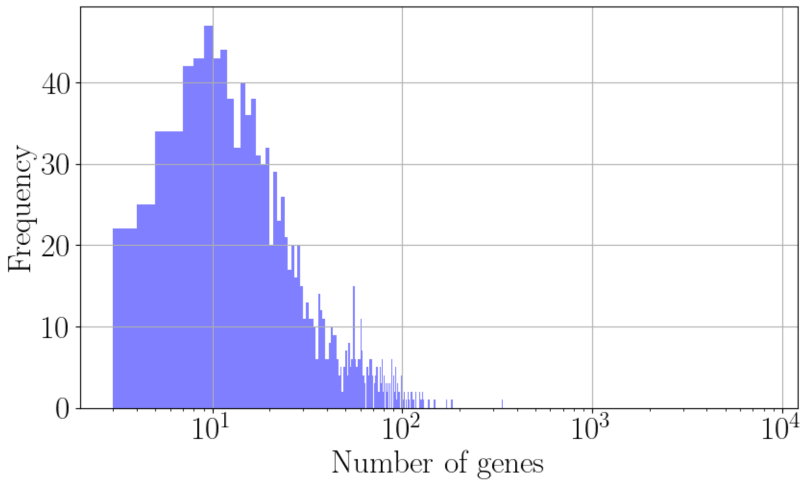


Figure 3.6: Histogram of the number of genes for each of the 1233 pathways

2.2.1 Unmapped genes: spectral clustering

Notwithstanding the subdivision of the graph according to gene pathways, the Unmapped group consists of 8,070 genes, hence the size of the x-axis in Figure 3.6. However, due to the log-scale, the width of the bin representing the Unmapped group is too thin to perceive. We perform spectral clusterings in an iterative manner to reduce the size of this group into clusters of a manageable size.

Spectral clustering is a technique for community detection in a graph. Let A be the symmetric, $|V| \times |V|$ adjacency matrix of the unweighted graph. The symmetric normalised Laplacian is defined as $L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. This method computes the eigenvectors corresponding to the smallest k eigenvalues of the Laplacian, excluding the trivial one $\lambda_1 = 0$ which always exists. These eigenvectors form a lower-dimensional representation of the data, where k-means clustering is used to separate the data in the reduced spectral space into k different clusters [49, 50]. Let λ_i denote the eigenvalues of the normalised Laplacian, such that $|\lambda_{|V|}| \geq |\lambda_{|V|-1}| \geq \ldots \geq |\lambda_1|$, with $\lambda_1 = 0$ being the trivial solution. If the largest eigengap occurs for $|\lambda_{k+1} - \lambda_k|$, then k is often considered to be the optimal number of clusters [51].

Spectral clustering is well suited for identifying complex relationships between the nodes, which is particularly useful given our goal of describing higher-order interactions. Unlike simple clustering algorithms that capture first-order relationships, spectral clustering reflects more intricate interactions among nodes. Moreover, the iterative approach allows to progressively subdivide the larger clusters into smaller ones while considering the structure and optimal number of subclusters at each step of the process.

We use the implementation of this algorithm from the *sklearn* library for Python [52]. Before the first iteration, we observe that the smallest two eigenvalues from the adjacency matrix of the Unmapped subgraph are both 0, meaning that it has two disconnected components. Using the *networkx* library [53], we find that 8,069 nodes are connected in one single component, while the node representing the IQCM gene is isolated, so we remove it from the subgraph. The smallest 20 eigenvalues of the new adjacency matrix are shown in Figure 3.7. The largest eigengap after excluding the smallest, trivial eigenvalue $\lambda_1 = 0$ occurs between λ_3 and λ_4, implying that the best number of clusters for the first iteration is 3. These 3 clusters are shown by colour in Figure 3.8(b). Performing this process recursively until all the resulting clusters have a size that allows for their maximal clique initialisation results in 37 different node clusters as shown in Figure 3.8(c), giving a total of 1,268 subgraphs with the rest of the pathways.

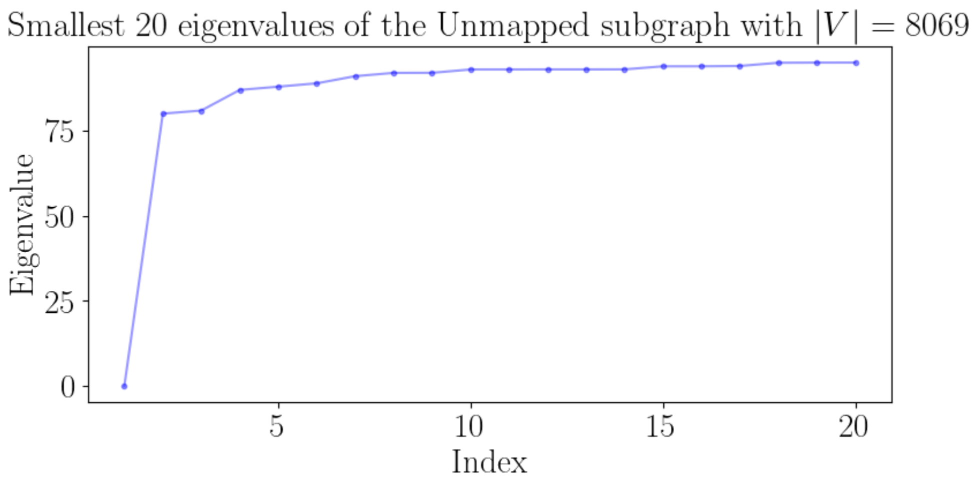
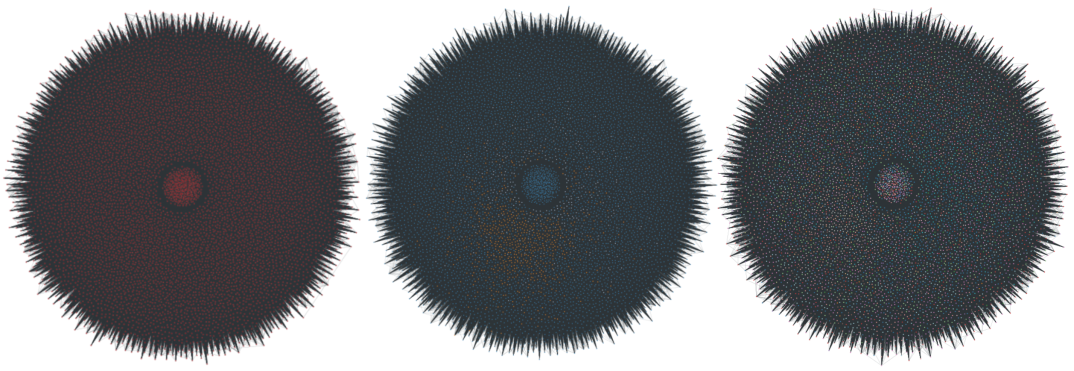


Figure 3.7: Smallest 20 eigenvalues of the Unmapped adjacency matrix after removing the only disconnected node.



(a) Unmapped group (b) First 3 clusters (c) 37 final clusters

Figure 3.8: Iterative spectral clustering for the Unmapped group. Left (a): unmapped nodes; centre (b): unmapped nodes coloured by belonging to each of the 3 clusters obtained in the first iteration; right (c): unmapped nodes coloured by belonging to each of the final 37 clusters.

2.3 Implementation

In order to infer the corresponding hypergraph from each of the 1,268 subgraphs using *graph-tool*'s implementation of the method proposed by Young et al. [17], we first perform the maximal clique initialisation, possible thanks to the spectral clustering, and then explore the posterior distribution using the MCMC random walk. When using MCMC random walks, the *burn-in period* refers to an initial, exploratory phase where the algorithm progressively converges to the target posterior distribution, $P(H|G)$ in our case. The purpose of this period is to discard the first iterations, influenced by the initial state, to ensure convergence to the posterior distribution [54].

We perform a burn-in of 10,000 sweeps for subgraphs with an order lower than 500. After that, the already-converged algorithm explores 1,000 different hypergraphs drawn from the posterior distribution $P(H|G)$ and selects the one with highest posterior probability, as defined in [17].

However, 6 subgraphs with more than 500 nodes still pose a computational challenge: every step of the MCMC walk takes around 20 minutes. We decide to skip the burn-in period for these ones and select the hypergraph with the highest posterior probability from the first 1,000 iterations, which take around 2 weeks of computations. Even during the first exploratory iterations, the MCMC model can find a state with a high posterior probability, indicating that the chain is close to a favorable region of the state space. Since the measure of posterior probability used by Young et al. [17], the *entropy* of the hypergraph, correlates with the quality of the hyperedge partitioning, said state is likely to be a good approximation of the true posterior distribution, according to the principle of parsimony. Still, omitting the burn-in period does introduce the risk of a bias.

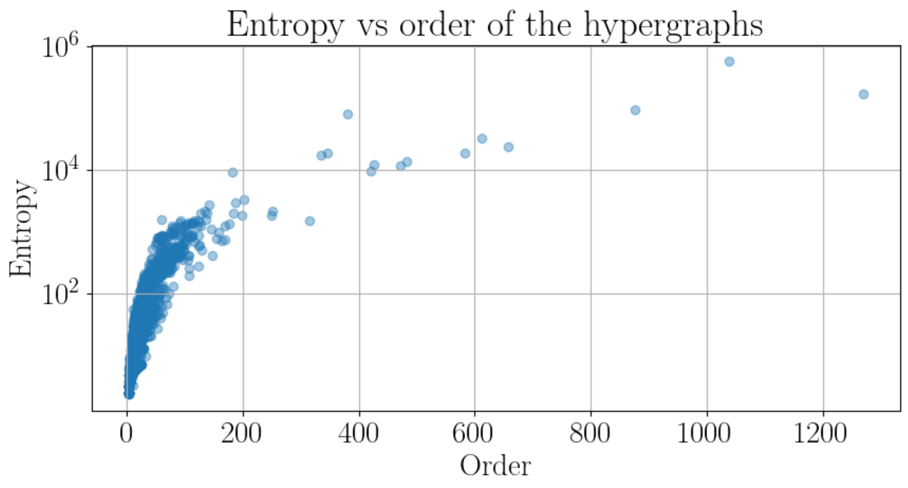


Figure 3.9: Lowest entropy obtained for each hypergraph as a function of its order.

Figure 3.9 shows the lowest entropy obtained for each of the 1,268 subgraphs. The entropy of a hypergraph in this algorithm is introduced in [17], where a lower entropy implies a higher posterior probability. Despite the risk introduced by omitting the burn-in period on the six hypergraphs with largest order, they seem to follow the trend shown by the rest. We expect the entropy to increase exponentially with hypergraph order. Figure 3.9 shows a sub-exponential dependency, meaning that skipping the burn-in period for the largest hypergraphs does not pose a significant disadvantage.

3 Feature extraction for ML

In this section, we explain the processes followed to obtain both sets of features for the ML classification models: the topological metrics and the node embeddings. For a proper comparison with the 1,268 hypergraphs, the graph obtained in section 1 is also clustered into the same number of subgraphs. This subdivision means that the ML features will be generated not for each gene individually, but rather for each node within each subgraph (i.e., gene-pathway pairs). Hence, a gene will be classified as essential or not for every pathway it is involved in.

3.1 Topological metrics

All the metrics described in chapter 2, section 3 will be used as features for the ML models. Hence, the feature matrix for this approach will have as many rows as gene-pathway pairs, where each row will display the degree, betweenness, closeness, and eigenvector centrality metrics, together with the clustering and bottleneck coefficients.

The features for the graph representation can be obtained using the algorithms implemented in the *networkx* library. Only the bottleneck coefficient is not implemented, for which we use the pseudo-code from Algorithm 2.

3.1.1 Hypergraph metrics

The representations and mappings reviewed in chapter 2, section 1.2 allow the topological metrics to be well-defined for a hypergraph. We will use the star expansion of our hypergraphs.

Every representation choice comes with some disadvantages. The star expansion can increase the complexity of the hypergraph, as the number of nodes and edges can increase. Also, two nodes from the hypergraph (genes) never share a direct connection in the star expansion. Only connections through the other set of nodes (hyperedges) exist, which can obscure some interactions between genes. Nevertheless, compared to the other choices and their disadvantages, the use of the star expansion is most appropriate. The clique expansion loses information about the higher-order interactions, while the motifs and simplicial complexes grow exponentially in computational complexity with respect to the number of nodes. Thus, even after the pathway clustering described in section 2, those clusters with hundreds of nodes would pose a challenge. Star expansions store all the higher-order information and are computationally manageable.

Algorithm 2 Computation of the bottleneck coefficient.

```
Require: networkx Graph object representing a certain subgraph $G = (V, E)$, or
    star expansion representing a certain hypergraph $G_* = (V_* = V \cup E, E_*)$.
Ensure: $G$ (or $G_*$) has no disconnected components.
 1: $n \leftarrow |V|$.    ▷ if star expansion, only consider nodes in $V$, not all nodes in $V_*$
 2: Initialise bottleneck as an empty dictionary.
 3: for node $v$ in $V$ do
 4:     Initialise path_counts as an empty dictionary.
 5:     $T \leftarrow$ shortest path tree rooted at $v$ as a dictionary where the keys are the
        nodes from $G$ and the values are lists with the shortest paths from $v$ to the
        corresponding node (built-in in networkx).
 6:     for target $t$, path in $T$ do
 7:         if $t = v$ then
 8:             Skip this iteration.
 9:         end if
10:         for node $i$ in path do
11:             if $i \neq v$ and $i \neq t$ and $i \in V$ then
12:                 path_counts $[i] \leftarrow$ path_counts $[i] + 1$.
13:             end if
14:         end for
15:     end for
16:     for node $i$, count in path_counts do
17:         if count $> n/4$ then
18:             bottleneck $[i] \leftarrow$ bottleneck $[i] + 1$.
19:         end if
20:     end for
21: end for
```

Let us now describe the computation of each of the topological metrics for the bipartite graphs $G_* = (V_* = V \cup E, E_*)$, where V is the set of nodes in the hypergraph and E the set of hyperedges. Firstly, degree centrality is straightforward to compute using the library *hypergraphx* and its built-in functions [55].

The computation of the betweenness centrality is not as straightforward. Using the *networkx* built-in function on the star expansion returns a value for each node of the star expansion in V_*. Selecting the values corresponding to original nodes V from the hypergraph while dropping those corresponding to hyperedges E is not enough, as the betweenness centrality of every node depends on the shortest paths from all sources to all targets. In our star expansion, this means that the betweenness centrality of a node also counts shortest paths from hyperedges to nodes, and from hyperedges to hyperedges. Hence, the computation needs to account only for the nodes of the star expansion that correspond to nodes in the hypergraph. In mathematical terms, the summation in Definition 2.10 for the betweenness centrality of node v should not go over all nodes s and t such that $s \neq v \neq t$, but rather over all nodes s and t such that $s \neq v \neq t : s, t \in V$.

Similarly, care needs to be taken with the closeness centrality. The summation in Definition 2.11 needs to be adapted so that instead of considering all nodes u such that $u \neq v$, it considers only the nodes u such that $u \neq v : u \in V$. The implementation of the betweenness and closeness computations for the star expansions are similar to the for-loop in line 3 from Algorithm 2, where the computation is only performed for the original hypergraph nodes. Algorithms 3 and 4 from appendix B show the details of each implementation.

For the eigenvector centrality, we can compute the dominant eigenvector and select only the first $|V|$ entries for each hypergraph, dropping the entries corresponding to the nodes that represent hyperedges. Therefore, the power method can be used, as is done for the graphs. The maximum number of iterations is set to 10,000, and the tolerance to 10^{-8}.

The definition of the clustering coefficient is not applicable to hypergraphs. Thus, we propose defining the clustering coefficient for a node in a hypergraph as the ratio of the number of hyperedges it belongs to over the number of all possible hyperedges between the node and its neighbours. The number of hyperedges containing a node is straightforward to obtain with *hypergraphx*, and the number of all possible hyperedges between a node and its n neighbours is given by $2^{n+1} - n - 2$. The term 2^{n+1} is the cardinality of the power set of the set containing the node and its n neighbours. We deduct $n + 2$ hyperedges, one being the empty set, and the other $n + 1$ being the hyperedges containing a single node.

Finally, the bottleneck computation follows Algorithm 2, which only iterates through sources and targets in V, but not through nodes representing hyperedges.

3.2 Embeddings

We embed the graph nodes using the *node2vec* algorithm [35]. In order to implement it, a *networkx* Graph can be used as input for each subgraph. Table 3.1 displays the parameter values used as inputs.

Table 3.1: Parameters used for the instantiation of the *Node2Vec* class from the *node2vec* library.

Parameter	**Value(s)**
Number of dimensions	[5, 10, 40, 128]
Random walk length	30 nodes
Number of random walks	200

We use the Hypergraph Networks with Hyperedge Neurons (HNHN) [38] algorithm to obtain node embeddings for the hypergraphs. Table 3.2 shows the values of the parameters used to compute the embeddings. The *dropout probability* is the probability with which neurons in the hypergraph NN are temporarily removed during the training of the model to prevent overfitting. The *edge predictions* parameter is set to *False* because we only want the node embeddings.

Table 3.2: Parameters used for the instantiation of the *Hypergraph* class from the *HNHN* library.

Parameter	**Value(s)**
Number of dimensions	[5, 10, 40, 128]
Number of hidden layers	2
Dropout probability	0.3
Edge and node weight initialisation	All set to 1
Edge predictions	False

4 ML algorithms

We use the supervised ML algorithms presented in chapter 2, section 5 with labels provided by the Inferred Common Essentiality dataset from DepMap [12], as introduced in section 2 from the same chapter. Recall that the inputs for these models are gene-pathway pairs, given the pathway subdivision for the hypergraph reconstruction.

4.1 Model selection

Each model is evaluated using both graph-based and hypergraph-based features. Specifically, five distinct sets of features are extracted from the graph and five from the hypergraph. These five sets correspond to the topological metrics and the embeddings into 5-, 10-, 40-, and 128-dimensional vector spaces (see Tables 3.1 and 3.2). For each model shown in Table 3.3, we conduct a hyperparameter search using *sklearn*'s GridSearchCV. This process explores various hyperparameter combinations to identify the best-performing set on an 80% training subset of the gene-pathway pairs, using 5-fold cross validation. The hyperparameters explored for each model are detailed in Table 3.3. The role of each hyperparameter can be found in [52].

Table 3.3: Hyperparameters explored for each algorithm

Model	**Hyperparameter**	**Values**
Logistic Regression	Solver	[lbfgs, liblinear, newton-cholesky]
	C	[0.1, 1, 10]
	Max. iterations	[5000]
Random Forest	Number of trees	[50, 100, 200, 300, 400, 500]
	Max Depth	[None, 10, 20, 30]
Neural Network	Neurons per hidden layer	[(50), (100), (100, 50), (100, 75, 50)]
	Activation	[relu]
	Learning Rate	[0.001]
Support Vector Classifier	Kernel	[poly, rbf]
	C	[0.1, 1, 10]
	Gamma	[scale]
K-Nearest Neighbours	Number of neighbours	[1, 3, 5, 7, 10, 15]

From the results on the 20% test set, the model achieving a higher accuracy among those using embeddings, regardless of the number of dimensions, and the model with a highest accuracy using topological metrics will be selected for further analysis. This selection process is applied separately to the graph and hypergraph models, resulting in four ML models being chosen – two from the graph-based features and two from the hypergraph-based features.

Each for the graph and the hypergraph, the two selected models will be compared and one will be chosen, resulting in only one model from each representation (graph/hypergraph). In order to make this comparison, both the precision and recall metrics for the "essential genes" class are useful. A high precision implies few false positives, meaning that those genes classified as essential probably are essential; a high recall means few false negatives, capturing most of the actual essential genes. Overlooking an essential gene could mean omitting a critical target for cancer therapy. The opposite is also undesirable but might be less critical than missing an essential gene. Thus we prioritise a high recall for the essential class, ensuring the model identifies as many essential genes as possible.

4.2 Hypergraph resemblance with CORUM complexes

In order to analyse if the hyperedges from the constructed hypergraphs represent real interactions, we will compare the hyperedges associated with the most confident predictions to the CORUM database of human protein complexes [18]. Specifically, we select the gene-pathway pairs from the test set that are classified with a confidence score of at least 0.8 by the best-performing hypergraph-based ML model. For each pair, the hyperedges containing the gene in the hypergraph associated with the corresponding pathway will be checked against the CORUM dataset. The number of complexes found as a subset of those hyperedges will be obtained.

4.3 Pathway-specific essentiality

The Common Inferred Essentiality dataset from DepMap [12] only labels each gene as either essential or non-essential, without the possibility of multiple labels depending on the pathway. Given the need for pathway subdivision due to computational cost and the research done on context-specific essentiality [27], we expect the ML models to classify some genes as essential in certain pathways and as non-essential in others.

With the aim of finding genes that potentially have pathway-specific essentiality, we explore those gene-pathway pairs that are classified by our best-performing models with a confidence score of at least 0.8. We group these high-confidence pairs by gene, and compute for every gene the *essentiality ratio* as n_1/n, where n_1 is the number of pathways in which that gene is classified as essential and n is the total number of gene-pathway pairs featuring that gene. Those genes with an essentiality ratio of 0.5 have been classified as essential in the same number of pathways as non-essential. We select the genes with an essentiality ratio in the interval $[0.1, 0.9]$. The resulting genes will have been classified with a high confidence both as essential and non-essential in different pathways, thus possibly displaying pathway-specific essentiality.

4 Results and analysis

1 Graph models

In this section we compare the models trained on topological metrics from the subgraphs to the models taking node embeddings as features. We first select the best model trained on topological metrics from within all the models in Table 3.3. Likewise, the models trained on node embeddings are explored to find the best-performing one. Finally, one single model is selected as the graph-based best-performing model.

1.1 Topological metrics

Table 4.1 displays the optimal hyperparameters and the respective accuracy obtained for each model that used topological metrics from the subgraphs. The model that is selected for analysis is the RF with 300 trees and no maximum depth.

Table 4.1: Best-performing hyperparameters and accuracy for each model trained on topological metrics from the subgraphs.

Model	**Best hyperparameters**	**Accuracy**
LR	C = 10, solver: lbfgs	0.722
RF	**Number of trees: 300, Max. depth: None**	**0.820**
NN	2 layers: (100, 100)	0.812
SVC	C = 1, kernel: poly	0.739
KNN	Number of neighbours: 10	0.806

1.2 Embeddings

Table 4.2 shows the optimal hyperparameters, the number of dimensions, and the accuracy obtained for each model that used embeddings from the subgraphs. We select KNN with $k = 3$ for the embeddings on a 40-dimensional vector space.

Table 4.2: Best-performing hyperparameters, embedding dimensionality, and accuracy for each model trained on embeddings from the subgraphs.

Model	**Number of embedding dimensions**	**Best hyperparameters**	**Accuracy**
LR	5	C = 1, solver: liblinear	0.689
RF	10	$\begin{cases}\text{Number of trees: 400} \\ \text{Max. depth: None}\end{cases}$	0.791
NN	128	2 layers: (100, 50)	0.797
SVC	128	C = 10, kernel: rbf	0.825
KNN	**40**	**Number of neighbours: 3**	**0.833**

1.3 Best graph model

The performance statistics for both selected models taking graph-based features are displayed in Tables 4.3 and 4.4.

Table 4.3: Classification metrics: RF trained on topological subgraph metrics.

Metric	**Non-essential**	**Essential**	**Overall**
Accuracy			0.820
Precision	0.85	0.69	
Recall	0.91	0.57	
F1-Score	0.88	0.62	
Support	6963	2486	
Confusion matrix:	Predicted non-essential	Predicted essential	
True non-essential	6339	624	
True essential	1081	1405	

Table 4.4: Classification metrics: KNN trained on subgraph node embeddings.

Metric	**Non-essential**	**Essential**	**Overall**
Accuracy			0.833
Precision	0.85	0.75	
Recall	0.94	0.55	
F1-Score	0.89	0.63	
Support	6963	2486	
Confusion matrix:	Predicted non-essential	Predicted essential	
True non-essential	6515	448	
True essential	1128	1358	

From these two best-performing graph-based models, the RF with topological metrics as features has a higher recall metric for the essential class. In the KNN model, 3 out of 4 genes classified as essential are true positives. However, it only identifies a 55% of all the essential genes, while the RF identifies a 57% of them. Hence, we select the RF using subgraph topological metrics, as it is able to correctly identify more essential genes.

2 Hypergraph models

In this section we compare the models trained on topological hypergraph metrics to the models trained on node embeddings. We first select the best model trained on topological metrics. Similarly, the models trained on node embeddings are explored to find the best-performing one. Then, one single model is selected as the hypergraph-based best-performing model. Finally, the most confident predictions from the selected model will be explored and checked against Core CORUM protein complexes [18].

2.1 Topological metrics

Table 4.5 shows the optimal hyperparameters and the accuracy obtained for each model that used topological metrics from the hypergraphs. The best-performing model is the RF, this instance with 500 trees and without a maximum depth.

Table 4.5: Best-performing hyperparameters and accuracy for each model trained on topological metrics from the hypergraphs.

Model	**Best hyperparameters**	**Accuracy**
LR	C = 10, solver: lbfgs	0.671
RF	**Number of trees: 500, Max. depth: None**	**0.819**
NN	2 layers: (100, 100)	0.798
SVC	C = 10, kernel: poly	0.737
KNN	Number of neighbours: 10	0.791

2.2 Embeddings

Table 4.6 shows the optimal hyperparameters, the number of dimensions, and the accuracy obtained for each model that used embeddings from the hypergraphs. The best-performing model is the RF for the 128-dimensional embeddings, with 400 trees and no maximum depth.

Table 4.6: Best-performing hyperparameters, embedding dimensionality, and accuracy for each model trained on embeddings from the hypergraphs.

Model	**Number of embedding dimensions**	**Best hyperparameters**	**Accuracy**
LR	128	C = 1, solver: lbfgs	0.577
RF	**128**	**Number of trees: 400, Max. depth: None**	**0.837**
NN	10	2 layers: (100, 100)	0.744
SVC	40	C = 10, kernel: rbf	0.768
KNN	128	Number of neighbours: 10	0.823

2.3 Best hypergraph model

The performance statistics for both selected models taking hypergraph-based features as inputs are displayed in Tables 4.7 and 4.8.

The model using hypergraph 128-dimensional embeddings from Table 4.8 achieves a slightly higher accuracy than the one using topological metrics. Still, the recall and F1-score are higher for the model trained on topological metrics, as shown in Table 4.7. Hence, this model finds more essential genes, as can also be confirmed by the confusion matrices. Like for the graph-based features, the topological hypergraph

Table 4.7: Classification metrics: RF trained on topological hypergraph metrics.

Metric	**Non-essential**	**Essential**	**Overall**
Accuracy			0.819
Precision	0.85	0.71	
Recall	0.92	0.54	
F1-Score	0.88	0.61	
Support	6963	2486	
Confusion matrix:	Predicted non-essential	Predicted essential	
True non-essential	6409	554	
True essential	1153	1333	

Table 4.8: Classification metrics: RF trained on hypergraph node embeddings.

Metric	**Non-essential**	**Essential**	**Overall**
Accuracy			0.837
Precision	0.84	0.85	
Recall	0.97	0.47	
F1-Score	0.90	0.60	
Support	6963	2486	
Confusion matrix:	Predicted non-essential	Predicted essential	
True non-essential	6755	208	
True essential	1330	1156	

metrics show better performance for our purposes than the node embeddings. Thus, the optimal hypergraph-based model is the RF trained on topological metrics.

2.4 Hypergraph resemblance with CORUM complexes

As specified in chapter 3, section 4, the hyperedges associated with the most reliable gene-pathway pair predictions are checked against the Core CORUM database. The reliability of a RF prediction for a given input can be measured by the fraction of trees that would classify the input to each of the classes. In our case, this means computing the fraction p of trees that would classify a given input from the test set as essential. In order to keep the most confident predictions without disregarding the non-essential ones, if $p < 0.5$ then we redefine it as the fraction of trees that would classify the input as non-essential. Effectively, we can re-assign its value as $p \leftarrow 1 - p$ in those cases.

From within the test set containing 9,449 gene-pathway pairs, there are 3,644 pairs classified consistently by at least an 80% of the 500 trees, i.e., with a confidence score greater or equal to 0.8. For each of those gene-pathway pairs we select the hyperedges from the corresponding pathway that contain the gene and check whether there is a CORUM complex as a subset of any of those hyperedges. Table 4.9 shows the results of this analysis. The cell lines or tissue types associated with the 77 found complexes are listed in Table C.1.

Table 4.9: Analysis of the gene-pathway pairs from the test set in the selected hypergraph-based RF.

Number of pairs with confidence score ≥ 0.8	3644
Number of distinct pathways within those pairs	773
Number of pathways with some hyperedge containing a CORUM complex	108
Number of distinct complexes found as subsets of those hyperedges	77

3 Pathway-specific essentiality

Figure 4.1 shows the genes with a confidence score above 0.8 whose essentiality ratio is in the interval $[0.1, 0.9]$. However, we limit the visualisation to only those genes who are labelled as non-essential in the Inferred Common Essentiality dataset from DepMap. We make this restriction to find genes that were unknown to be essential in certain pathways. Figure 4.1(left) shows the genes obtained from the RF with 300 trees using topological graph metrics, while Figure 4.1(right) shows the genes obtained for the RF with 500 trees using topological hypergraph metrics.

Table 4.10 shows all the non-essential genes from the DepMap database that are classified as essential in certain pathways by our graph- or hypergraph-based models. Note that the genes AGO3, AGO4, and UBC are classified as essential at least once by both models. More specifically, the genes AGO3 and AGO4 are classified as essential in the same pathway by both models, which suggests that they indeed behave as essential in the pathway R-HSA-8934593.

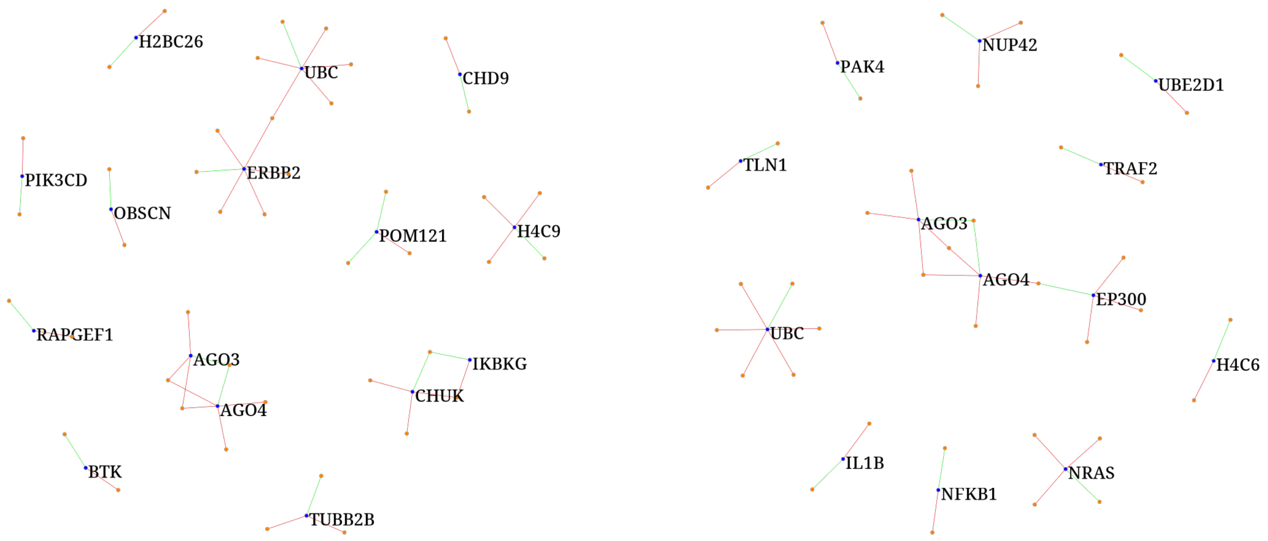


Figure 4.1: High-confidence predictions for the RF model trained on topological graph metrics (left) and on topological hypergraph metrics (right). Only genes classified as non-essential in DepMap are shown. Blue nodes represent genes, orange nodes represent pathways, and edges represent gene-pathway pairs. Green edges are pairs predicted as essential; red edges show otherwise.

4 Discussion

4.1 ML algorithm selection

All models tend to classify most genes as non-essential. This trend is explained by the class imbalance: only around 1/4 of the gene-pathway pairs are labelled as essential in the Inferred Common Essentiality dataset. Despite using class weights for the LR, SVC, and RF models, class imbalance biases models toward the majority class: non-essential genes. This effect is slightly more significant in KNN and NN as they do not account for class imbalance.

The two selected models, with performance metrics shown in Tables 4.3 and 4.7, show similar efficacy in predicting essential genes. The graph-based model correctly identifies 72 more essential genes than the hypergraph-based model, hence the slightly higher recall score for the essential class. However, the precision score for the essential class is higher in the hypergraph-based model, resulting in almost equal F1-scores for both models.

Therefore, the hypergraph representation does not yield significantly improved

Table 4.10: Pathway-specific essentialities obtained for genes currently considered non-essential.

Graph-based model		**Hypergraph-based model**	
Gene	**Essential pathways**	**Gene**	**Essential pathways**
AGO3	R-HSA-8934593	AGO3	R-HSA-8934593
AGO4	R-HSA-8934593	AGO4	R-HSA-8934593
BTK	R-HSA-5663213	EP300	R-HSA-8936459
CHD9	R-HSA-2151201	H4C6	R-HSA-4551638
CHUK	R-HSA-1236974	IL1B	R-HSA-9020702
ERBB2	R-HSA-1257604	NFKB1	R-HSA-933542
H2BC26	R-HSA-2559586	NRAS	R-HSA-9607240
H4C9	R-HSA-3214841	NUP42	R-HSA-159236
IKBKG	R-HSA-1236974	PAK4	R-HSA-9013423
OBSCN	R-HSA-9013406	TLN1	R-HSA-5674135
PIK3CD	R-HSA-2219530	TRAF2	R-HSA-5668541
POM121	R-HSA-159236 R-HSA-9615933	UBC	R-HSA-8948751
RAPGEF1	R-HSA-912631	UBE2D1	R-HSA-5357905
TUBB2B	R-HSA-389977		
UBC	R-HSA-983168		

predictions for gene essentiality. In spite of simplifications such as the pathway clustering, the further iterative spectral clustering for the unmapped genes, and the omission of the burn-in period for the MCMC algorithm of the six largest clusters, the performance of the graph- and hypergraph-based models are remarkably similar. This suggests that improved hypergraph-reconstruction models where said simplifications are not needed could produce hypergraphs with more predictive power. Still, given the extensive time and computational resources required to construct the hypergraph – enumerating maximal cliques from the given graph and performing the MCMC walk –, the cost-benefit ratio for an example with as many nodes and interactions could favour graph-based approaches when computational resources are limited.

The topological metrics are more useful for node prediction both in the graph- and hypergraph-based models in comparison to the embeddings. Often, node embeddings compress information about the nodes and their neighbourhoods into dense, high-dimensional vectors, which can lead to overfitting the model [56]. Despite their ability to capture complex relationships, they do not seem to offer additional information beyond what topological metrics offer. Topological metrics are known to be important for classification tasks in biological networks [15], while embedding algorithms such as *node2vec* or *HNHN* do not seem to perform as well in biological contexts.

4.2 Hypergraph resemblance with CORUM complexes

The most reliable predictions from the hypergraph-based model are associated to pathways whose hyperedges include 77 distinct CORUM complexes as subsets. This observation suggests that the method to reconstruct higher-order data from pairwise interactions [17] effectively captures biologically meaningful relationships at the protein complex level, even despite the simplifications made in the process. The fact that CORUM complexes, well-characterised groups of interacting proteins, are found within the hyperedges of the most reliable predictions highlights the usefulness of hypergraphs to model complex biological systems with higher-order interactions.

4.3 Pathway-specific essentiality

Given the class imbalance in the training dataset, where approximately 75% of the entries correspond to non-essential genes, the algorithm is prone to misclassifying essential genes as non-essential. As a result, the essential genes obtained in this analysis (not included in Figure 4.1 or Table 4.10) might have been incorrectly labelled as non-essential in certain pathways by the model. The model may have learned to prioritise the more frequent non-essential class, reducing its sensitivity to essential genes. However, it is also possible that our ML model finds genes usually considered essential behaving as non-essential for certain pathways.

In contrast, the genes identified as non-essential by DepMap but as essential in some pathways by our graph- or hypergraph-based models, more likely represent true instances of pathway-specific essentiality. These genes, though generally considered non-essential, may play key roles in some pathways, exhibiting essentiality in a context-dependent manner [27]. We highlight the pathway-specific essentiality of the genes AGO3 and AGO4, involved in RNA binding and transcription.

5 Conclusions and future work

1 Conclusion

In this report, we studied the utility of graph and hypergraph descriptions for modelling gene co-dependency in cancer cell lines. Despite the complexity and computational intensity involved in constructing hypergraphs, including the necessity for simplifications such as pathway clustering, spectral clustering of unmapped genes, and short MCMC random walks, the resulting hypergraph-based models performed similarly to the graph-based models. The comparable performance of both models despite the various simplifications made during the hypergraph reconstruction suggests a strong predictive power for hypergraphs. If the hypergraph reconstruction method could have been applied to the entire graph without the need for clustering, or if the MCMC walks had been allowed to fully converge with an extensive burn-in period, it is likely that hypergraphs would have displayed even greater predictive accuracy. Therefore, while hypergraphs offer a richer description of biological interactions, their practical utility may require more precise and computationally efficient methods to fully exploit their potential.

Furthermore, the comparison with CORUM complexes – where hypergraph-based models successfully recovered 77 distinct complexes as subsets of their hyperedges – underscores the potential of hypergraphs to describe biological structures that are missed by simpler graph-based representations. Hence, hypergraphs offer a more detailed and biologically relevant description of gene interactions.

Finally, a study of the predictions with highest confidence score in the selected graph- and hypergraph-based ML models pinpoints genes that are classified as non-essential in the DepMap data but might display pathway-specific essentiality.

2 Future work

This study confirms that improvements and optimisations for the algorithms to recover hypergraphs from data are needed for real-world data. We found that the algorithm developed by Young et al. needs excessive computational resources for inferring gene dependency as a hypergraph. Still, we showed that the graph input data can be simplified and clustered in such a way that the hypergraphs yield results nearly as accurate as the graphs.

Several distinct CORUM complexes have been found as subsets of hyperedges from the most confident predictions, as discussed in section 4.2 from chapter 4. This finding has potential for further protein complex classification and discovery through the use of hypergraphs. It is possible that protein complexes yet undiscovered are subsets from the hyperedges associated with the most reliable hypergraph-based predictions. This prospect encourages the development of algorithms tailored to explore these hypergraph predictions. Such algorithms could use the structural information from the hypergraphs to identify novel protein complexes.

The genes shown in Figure 4.1, labelled as non-essential by the Inferred Common Essentiality dataset, can be potential genes showing unknown pathway-specific essentialities. Future work could involve experimental studies to confirm the pathway-specific roles of these genes, also displayed in Table 4.10, mainly concerning AGO3 and AGO4 given that both are predicted to be essential when expressed in the pathway R-HSA-8934593 by both the graph-based model and the hypergraph-based model.

Bibliography

[1] M. Newman, *Networks: An Introduction.* Oxford University Press, 2010, ISBN: 9780199206650. DOI: `10.1093/acprof:oso/9780199206650.001.0001`.

[2] W. Huber, V. J. Carey, L. Long, S. Falcon, and R. Gentleman, "Graphs in molecular biology," *BMC Bioinformatics*, vol. 8, no. S8, 2007.

[3] G. Petri *et al.*, "Homological scaffolds of brain functional networks," *J. R. Soc. Interface*, vol. 11, no. 20140873, 2014. DOI: `10.1098/rsif.2014.0873`.

[4] A. E. Sizemore, C. Giusti, A. Kahn, J. M. Vettel, R. F. Betzel, and D. S. Bassett, "Cliques and cavities in the human connectome," *Journal of Computational Neuroscience*, vol. 44, pp. 115–145, 1 2018. DOI: `10.1007/s10827-017-0672-6`.

[5] C. Berge, *Graphs and Hypergraphs.* North-Holland Publishing Company, 1973, ISBN: 9780444103994.

[6] Y. Gao, Z. Zhang, H. Lin, X. Zhao, S. Du, and C. Zou, "Hypergraph Learning: Methods and Practices," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 2548–2566, 5 2022.

[7] A. Bretto, *Hypergraph Theory: An Introduction* (Mathematical Engineering). Springer, 2013, ISBN: 978-3-319-00079-4. DOI: `10.1007/978-3-319-00080-0`.

[8] F. Battiston *et al.*, "Networks beyond pairwise interactions: Structure and dynamics," *Physics Reports*, vol. 874, pp. 1–92, 2020. DOI: `10.1016/j.physrep.2020.05.004`.

[9] R. L. Siegel, K. D. Miller, N. S. Wagle, and A. Jemal, "Cancer statistics, 2023," *CA: A cancer Journal for Clinicians*, vol. 73, pp. 17–48, 1 2023. DOI: `10.3322/caac.21763`.

[10] Cancer Research UK, "Cancer in the UK: Overview 2024," Cancer Research UK, Tech. Rep., 2024.

[11] A. Tsherniak *et al.*, "Defining a Cancer Dependency Map," *PubMed Central*, vol. 170, 564–576.e16, 3 2017. DOI: `10.1016/j.cell.2017.06.010`.

[12] DepMap, Broad, *Current DepMap Release data, including CRISPR Screens, PRISM Drug Screens, Copy Number, Mutation, Expression, and Fusions*, DepMap 23Q4 Public. Figshare+. Dataset, 2023. DOI: `10.25452/figshare.plus.24667905.v2`.

[13] P. Y. Lum *et al.*, "Extracting insights from the shape of complex data using topology," *Scientific Reports*, vol. 3, no. 1236, 2013. DOI: `10.1038/srep01236`.

[14] H. Masoomy, B. Askari, S. Tajik, A. K. Rizi, and G. R. Jafari, "Topological analaysis of interaction patterns in cancer-specific gene regulatory network: Persistent homology approach," *Scientific Reports*, vol. 11, no. 16414, 2021. DOI: `10.1038/s41598-021-94847-5`.

[15] I. Wu and X. Wang, "A novel approach to topological network analysis for the identification of metrics and signatures in non-small cell lung cancer," *Scientific Reports*, vol. 13, no. 8223, 2023. DOI: `10.1038/s41598-023-35165-w`.

[16] J. Pan *et al.*, "Interrogation of Mammalian Protein Complex Structure, Function, and Membership Using Genome-Scale Fitness Screens," *Cell Systems*, vol. 6, no. 5, 555–568.e7, 2018. DOI: `10.1016/j.cels.2018.04.011`.

[17] J.-G. Young, G. Petri, and T. P. Peixoto, "Hypergraph reconstruction from network data," *Communication Physics*, vol. 4, no. 135, 2021. DOI: `10.1038/s42005-021-00637-w`.

[18] G. Tsitsiridis *et al.*, "CORUM: the comprehensive resource of mammalian protein complexes-2022," *Nucleic Acids Res.*, vol. 51, pp. D539–D545, D1 2023. DOI: `10.1093/nar/gkac1015`.

[19] B. Nagle, V. Rödl, and M. Schacht, "The counting lemma for regular k-uniform hypergraphs," *Random Structures & Algorithms*, vol. 28, no. 2, pp. 113–179, 2006. DOI: `10.1002/rsa.20117`.

[20] L. Qi, J.-Y. Shao, and Q. Wang, "Regular uniform hypergraphs, s-cycles, s-paths and their largest Laplacian H-eigenvalues," *Linear algebra and its applications*, vol. 443, pp. 215–227, 2014. DOI: `10.1016/j.laa.2013.11.008`.

[21] M. T. Schaub, Y. Zhu, J.-B. Seby, T. M. Roddenberry, and S. Segarra, "Signal Processing on Higher-Order Networks: Livin' on the Edge... and Beyond," *Signal Processing*, vol. 187, p. 108 149, 2021. DOI: 10.1016/j.sigpro.2021.108149.

[22] F. Buekenhout and M. Parker, "The number of nets of the regular convex polytopes in dimension ≤ 4," *Discrete Mathematics*, vol. 186, pp. 69–94, 1 - 3 1998.

[23] D. Horak and J. Jost, "Spectra of combinatorial Laplace operators on simplicial complexes," *Advances in Mathematics*, vol. 244, pp. 303–336, 2013. DOI: 10.1016/j.aim.2013.05.007.

[24] H. Källberg *et al.*, "Gene-Gene and Gene-Environment Interactions Involving HLA-DRB1, PTPN22, and Smoking in Two Subsets of Rheumatoid Arthritis," *The American Journal of Human Genetics*, vol. 80, no. 5, pp. 867–875, 2007. DOI: 10.1086/516736.

[25] M. B. Taylor and I. M. Ehrenreich, "Higher-order genetic interactions and their contribution to complex traits," *Trends Genet.*, vol. 31, no. 1, pp. 34–40, 2015.

[26] J. A. Marsh, *Protein Complex Assembly: Methods and Protocols* (Methods in Molecular Biology, 1764). Springer New York, 2018, ISBN: 1-4939-7759-8.

[27] P. Cacheiro *et al.*, "Human and mouse essentiality screens as a resource for disease gene discovery," *Nature Communications*, vol. 11, pp. 1–11, 1 2020.

[28] M. Attene-Ramos, C. Austin, and M. Xia, "High Throughput Screening," in *Encyclopedia of Toxicology (Third Edition)*, P. Wexler, Ed., Oxford Academic Press, 2014, pp. 916–917, ISBN: 978-0-12-386455-0. DOI: 10.1016/B978-0-12-386454-3.00209-8.

[29] S. Wuchty and P. F. Stadler, "Centers of complex networks," *J. Theor. Biol.*, vol. 223, pp. 45–53, 1 2003. DOI: 10.1016/s0022-5193(03)00071-7.

[30] D. Koschützki and F. Schreiber, "Comparison of Centrality for Biological Networks," in *Proceedings of the German Conference on Bioinformatics (GCB 2004)*, Bielefeld, Germany: DBLP, Oct. 2004. [Online]. Available: https://dblp.org/rec/conf/gcb/KoschutzkiS04.

[31] O. Perron, "Zur Theorie der Matrices," *Mathematische Annalen*, vol. 64, pp. 248–263, 1907. DOI: 10.1007/BF01449896.

[32] G. Frobenius, "Ueber Matrizen aus nicht negativen Elementen," *Sitzungsberichte der Königlich Preussischen Akademie der Wissenschaften*, pp. 456–477, 1912.

[33] R. von Mises and H. Pollaczek-Geiringer, "Praktische verfahren der gleichungsauflösung," *Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 9, pp. 152–164, 1929. DOI: 10.1002/zamm.19290090206.

[34] C.-H. Chin, S.-H. Chen, H.-H. Wu, C.-W. Ho, M.-T. Ko, and C.-Y. Lin, "cyto-Hubba: identifying objects and sub-networks from complex interactome," *BMC Systems Biology*, vol. 8, no. S11, 4 2014. DOI: 10.1186/1752-0509-8-S4-S11.

[35] A. Grover and J. Leskovec, "node2vec: Scalable Feature Learning for Networks," *arXiv preprint arXiv:1607.00653*, 2016. DOI: 10.48550/arXiv.1607.00653.

[36] T. Mikolov, K. Chen, G. Corrado, and D. Jeffrey, "Efficient Estimation of Word Representations in Vector Space," *arXiv preprint arXiv:1301.3781v3*, 2013. DOI: 10.48550/arXiv.1301.3781.

[37] P. Leleux, S. Courtain, K. Françoisse, and M. Saerens, "Design of biased random walks on a graph with application to collaborative recommendation," *Physica A: Statistical Mechanics and its Applications*, vol. 590, p. 126 752, 2022. DOI: 10.1016/j.physa.2021.126752.

[38] Y. Dong, W. Sawin, and Y. Bengio, "HNHN: Hypergraph Networks with Hyperedge Neurons," *arXiv preprint arXiv:2006.12278v1*, 2020. DOI: 10.48550/arXiv.2006.12278.

[39] Z. Liu and J. Zhou, *Introduction to graph neural networks* (Synthesis lectures on artificial intelligence and machine learning ; #45). Springer, 2020, ISBN: 1681737663.

[40] P. H. Swain and H. Hauska, "The decision tree classifier: Design and potential," *IEEE transactions on geoscience electronics*, vol. 15, pp. 142–147, 3 1977. DOI: 10.1109/TGE.1977.6498972.

[41] E. Palmer and S. A.J., "On the number of trees in a random forest," *Journal of combinatorial theory*, vol. 27, pp. 109–121, 2 1979. DOI: 10.1016/0095-8956(79)90073-X.

[42] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *Journal of the Royal Statistical Society. Series B, Statistical methodology*, vol. 57, pp. 289–300, 1 1995.

[43] Y. Benjamini, "Discovering the false discovery rate," *Journal of the Royal Statistical Society. Series B, Statistical methodology*, vol. 72, no. 4, pp. 405–416, 2010.

[44] T. P. Peixoto, “The graph-tool python library,” *figshare*, 2014. DOI: 10.6084/m9.figshare.1164194.

[45] R. W. Darling and J. R. Norris, “Structure of large random hypergraphs,” *Ann. Appl. Probab.*, vol. 15, no. 1A, pp. 125–152, 2005. DOI: 10.1214/105051604000000567.

[46] D. F. Pinney, S. H. Pearson-White, S. F. Konieczny, K. E. Latham, and C. P. Emerson, “Myogenic lineage determination and differentiation: Evidence for a regulatory gene pathway,” *Cell*, vol. 53, no. 5, pp. 781–793, 1988. DOI: 10.1016/0092-8674(88)90095-5.

[47] T. Hämälä, M. J. Guiltinan, J. H. Marden, S. N. Maximova, C. W. dePamphilis, and P. Tiffin, “Gene Expression Modularity Reveals Footprints of Polygenic Adaptation in Theobroma cacao,” *Molecular Biology and Evolution*, vol. 37, pp. 110–123, 1 2019. DOI: 10.1093/molbev/msz206.

[48] M. Milacic *et al.*, “The Reactome Pathway Knowledgebase 2024,” *Nucleic Acids Research*, vol. 52, no. D1, pp. D672–D678, 2023. DOI: 10.1093/nar/gkad1025.

[49] F. Liu, D. Choi, L. Xie, and K. Roeder, “Global spectral clustering in dynamic networks,” *Proceedings of the National Academy of Sciences - PNAS*, vol. 115, no. 5, pp. 927–932, 2018.

[50] C. M. Le and E. Levina, “Estimating the number of communities in networks by spectral methods,” *arXiv preprint arXiv:1507.00827*, 2019. DOI: 10.48550/arXiv.1507.00827.

[51] U. von Luxburg, “A tutorial on spectral clustering,” *Statistics and Computing*, vol. 17, pp. 395–416, 2007. DOI: 10.1007/s11222-007-9033-z.

[52] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[53] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring network structure, dynamics, and function using NetworkX,” *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pp. 11–15, 2008.

[54] C. Robert and G. Casella, *Monte Carlo Statistical Methods*, 2nd ed. 2004. Springer New York, 2004, ISBN: 1-4757-4145-6.

[55] Q. F. Lotito *et al.*, “Hypergraphx: A library for higher-order network analysis,” *Journal of Complex Networks*, vol. 11, no. 3, 2023, ISSN: 2051-1329. DOI: 10.1093/comnet/cnad019.

[56] A. Dehghan-Kooshkghazi, B. Kamiński, Ł. Kraiński, P. Prałat, and F. Théberge, "Evaluating node embeddings of complex networks," *Journal of Complex Networks*, vol. 10, no. 4, cnac030, 2022. DOI: `10.1093/comnet/cnac030`.

A Cell lines of Core CORUM complexes selected for the graph reconstruction

T cell line ED40515	mucosal lymphocytes	CLL cells	monocytes	THP-1 cells
bone marrow-derived	monocytes, LPS-induced	THP1 cells	human blood serum	human blood plasma
plasma	CSF	human leukemic T cell JA3 cells	erythrocytes	peripheral blood mononuclear cells
African Americans	SKW 6.4	BJAB cells	Raji cells	HUT78
J16	H9	U-937	Jurkat T	NB4 cells
U937	early B-lineage	T-cell leukemia	lymphoblasts	whole blood and lymph
human neutrophil-differentiating HL-60 cells	human peripheral blood neutrophils	human neutrophils from fresh heparinized human peripheral blood	human peripheral blood	HCM
liver-hematopoietic	cerebral cortex	human brain	pancreatic islet	human hepatocyte carcinoma HepG2 cells
Neurophils	H295R adrenocortical	frontal cortex	myometrium	vascular smooth muscle cells
Dendritic cells	intestinal epithelial	Primary dermal fibroblasts	HK2 proximal	brain pericytes
HepG2	HEK 293 cells, liver	normal human pancreatic duct epithelial	pancreatic ductal adenocarcinoma	OKH cells
cultured podocytes	renal glomeruli	VSMCs	differentiated HL-60 cells	SH-SY5Y cells
frontal and entorhinal cortex	SHSY-5Y cells	hippocampal HT22 cells	primary neurons	neurons
renal cortex membranes	Kidney epithelial cells	skeletal muscle cells	Skeletal muscle fibers	differentiated 3T3-L1
brain cortex	cortical and hippocampal areas	human H4 neuroglioma	Thalamus	HISM
pancreas	RCC4	C2C12 myotube	XXVI muscle	SH-SY5Y neuroblastoma
HCC1143	Hep-2	PANC-1	HEK293T cells	HEK-293 cells
heart	epithelium	kidney	heart muscle	central nervous system
COS-7 cells	ciliary ganglion	striated muscle	PC12	293FR cells

Table A.1: List of keywords for the cell lines of Core CORUM protein complexes of interest. The complexes acting on any of these cell lines will have their statistical significance explored.

B Computation of betweenness and closeness node centrality for hypergraphs

Algorithm 3 Computation of the betweenness centrality for hypergraphs.

```
Require: networkx Graph object for the star expansion representing a certain hypergraph $G_* = (V_*, E_*)$.
Ensure: $G_*$ has no disconnected components.
 1: $n \leftarrow |V|$
 2: Initialise betweenness as an empty dictionary.
 3: for source $s$ in $V$ do
 4:     for target $t$ in $V$ do
 5:         all_shortest_paths ← list with all shortest paths between $s$ and $t$ where each element is a list of nodes (built-in in networkx).
 6:         for path in all_shortest_paths do
 7:             for node in path do
 8:                 if $node \neq s$ and $node \neq t$ and $node \in V$ then
 9:                     $betweenness\,[node] \leftarrow betweenness\,[node] + \frac{1}{\mathrm{len}(all\_shortest\_paths)}$.
10:                 end if
11:             end for
12:         end for
13:     end for
14: end for
15: for $v$ in $V$ do
16:     $betweenness\,[v] \leftarrow \dfrac{betweenness\,[v]}{(n-1)\,(n-2)\,/2}$
17: end for
```

Algorithm 4 Computation of the closeness centrality for hypergraphs.

```
Require: networkx Graph object for the star expansion representing a certain hypergraph $G_* = (V_*, E_*)$.
Ensure: $G_*$ has no disconnected components.
 1: Initialise closeness as an empty dictionary.
 2: for $v$ in $V$ do
 3:     path_lengths $\leftarrow$ dictionary of all the distances from $v$ to the rest of nodes (built-in in networkx).
 4:     $d_t \leftarrow 0$
 5:     for node in $V$ do
 6:         if $node \neq v$ then
 7:             $d_t \leftarrow d_t + path\_lengths\,[node]$
 8:         end if
 9:     end for
10:     $closeness\,[v] = \frac{|V| - 1}{d_t}$
11: end for
```

C Cell lines of Core CORUM complexes found as subsets of hyperedges

HeLa (S3) cells	HeLa (cell) nuclear extracts	HeLa mitochondria
SaOS-2 cells	U2-OS cells	HEK293(T) cells
HEK293 mitochondria	293 cell (lysates)	Jurkat cell extract
Jurkat 6 cells	NE2 cells	fibroblast GM3349 cells
E.coli	U937 cells	HL60 cells
primary skin fibroblasts	muscle mitochondria from patients	in vitro
SW480 cells	human DNA inserted into pQE-9 vector	MCF cells
MCF-7 cells	HepG2 cells	cerebral cortex
lymphoblasts	COS cells	placenta cells
brain	skin fibroblast cell lines	Transfected Cos7 cells

Table C.1: List of cell lines for the protein complexes found as subsets of hyperedges from the most confident predictions.