UNIVERSITY OF SOUTHAMPTON

Entropy Network Fusion

by

James E. Strudwick

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Social Sciences
School of Mathematical Sciences

February 2020

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF SOCIAL SCIENCES
SCHOOL OF MATHEMATICAL SCIENCES

Doctor of Philosophy

by James E. Strudwick

Thanks to the continual development of technology, a massive amount of data is now being produced on a daily basis. Because of this, new methods for analysis are needed, particularly ones that can analyse multiple datasets on the same set of objects. This is especially relevant in systems biology, where different datasets probe different aspects of the same underlying system. A popular and widely used method to analyse datasets is to transform the data into networks. Networks help visualise and quantify connections between samples, revealing structure and information that may not be visible at first, hence the popularity of their use, particularly in the analysis of biological systems. Entropy Network Fusion (ENF) is a new methodology for fusing, or combining, together multiple networks on the same set of objects (nodes) into one single output network. It works by finding a solution (network) whose clustering structure is as close as possible to the clustering structure of all the given input networks, using information-theoretic entropy as a guiding principle. ENF is designed with a level of generality, such that it is not restricted to any specific type of data, giving it a wide range of applications. We tested our methodology on five cancer sets and compared the performance to Similarity Network Fusion, a state-of-the-art network fusion algorithm. Whilst SNF may be a faster method, the output from ENF is significantly better in terms of performance. We then further developed an approximate version of our algorithm, approximate Entropy Network Fusion (aENF), which is significantly faster computationally for larger networks, further increasing its range of application.

Contents

List of Figures

List of Tables

Nomenclature

D_{ij}	The distance between vertex (patient) i and vertex j
μ	Scale factor
S_{ij}	The similarity between vertex (patient) i and vertex j
$^{(l)}$	An object associated with the l-th layer
$\{S^{(l)}\}$	A collection of similarity matrices
w_m	A weight for a given layer
$\{S^{(l)}, w_l\}$	A collection of similarity matrices paired with weightings for each layer
θ	The current output of ENF
t	The sum of all the entries of θ
$C(\{S^{(l)}\}, \theta)$	The ENF cost of the set $\{S^{(l)}\}$ at θ
$\widetilde{C}(\{S^{(l)}\}, \theta)$	The aENF cost of the set $\{S^{(l)}\}$ at θ
$H_{\mathcal{C}}(\_)$	The Shannon entropy of a probability distribution
$KL_{\mathcal{C}}(\_\|\_)$	The Kullback-Liebler divergence of two probability distributions
$JS_{\mathcal{C}}(\_, \_)$	The Jensen Shannon Divergence between two probability distributions
$H_{\mathcal{Q}}(\_)$	The Von Neumann entropy of a density matrix
$KL_{\mathcal{Q}}(\_\|\_)$	The Kullback-Liebler divergence of two density matrices
$JS_{\mathcal{Q}}(\_, \_)$	The Quantum Jensen Shannon Divergence between two density matrices
$H_{\mathcal{N}}(\_)$	The Von Neumann entropy of a network
$\widetilde{H_{\mathcal{N}}}(\_)$	The approximate Von Neumann entropy of a network
$JS_{\mathcal{N}}(\_, \_)$	The Quantum Jensen Shannon divergence of a network
$\widetilde{JS_{\mathcal{N}}}(\_, \_)$	The approximate Quantum Jensen Shannon divergence of a network
G	An adjacency matrix of a symmetric graph
D_G	The degree matrix of the graph G
$L(G)$	The Laplacian matrix of the graph G
$\bar{L}(G)$	The re-scaled Laplacian matrix of the graph G
λ_i	The i-th eigenvalue of $L(G)$
$\sigma(\_)$	The set of eigenvalues of a matrix
y_i, x_i	Left and right eigenvectors associated with the ith eigenvalue λ_i
$E(k, f)$	The elementary change matrix in the (k, f)-entry (and (f, k)-entry)
$E_{\mathcal{L}}(k, f)$	The Laplacian of an elementary change matrix
α	The learning rate used in gradient descent
γ	The gradient decay for the momentum term in gradient descent

aENF	approximate Entropy Network Fusion
CI	Concordance Index
CIA	Co-inertia Analysis
DI	Data Intergration
DPM	Dirichlet Process Mixture
EM	Expectation-Maximization
ENF	Entropy Network Fusion
FD	Finite Difference
GD	Gradient Descent
IR	Isoperimetric Ratio
IT	Information Theory
JSD	Jensen-Shannon divergence
KL	Kullback-Liebler divergence
KNN	K Nearest Neighbours
LRT	Log Rank Test
MCIA	Multiple Co-inertia Analysis
MDS	Multidimensional Scaling
MI	Mutual Information
MN	Multilayer Network
NAG	Nestrov accelerated gradient
NID	Normalised Information Distance
NVI	Normalised Variation of Information
PCA	Principal Component Analysis
POD	Percentage of Difference
PSDF	Patient-Specific Data Fusion
QJSD	Quantum Jensen-Shannon divergence
QM	Quantum Mechanics
SNF	Similarity Network Fusion
TMD	Transcription Module Discovery
wENF	weighted Entropy Network Fusion

Declaration of Authorship

I, James E. Strudwick declare that the thesis entitled *Entropy Network Fusion* and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- none of this work has been published before submission
- parts of this work have been published as:

Signed:.....

Date:.....

Acknowledgements

Whilst a PhD can only belong to one person, the title attached to only one name, it is by no means a solo effort. I would challenge anyone reading this to search history. Look through all academia's PhD students and I guarantee you none were done completely on their own with no support of any kind. The same is true for the one whose final work you hold in your hands now. Whilst I wish I could list every single person I want to thank, sadly, I fear that list would grow too big. Thus I shall try to restrain myself, but I do not make any promises.

First, and foremost, I would like to thank all my supervisors for their support during this process. Particularly to Ruben for his patience, thoughts, honest feedback and for reining me in when my intuition and excitement got ahead of me. Next I would like to thank everyone from both the level 7 and level 8 office for making every day enjoyable and all the sparks of ideas they set off. Particularly Fabio, Conrad and Kiko for their patience to listen to the various ideas I had, for not dismissing them and giving them their serious thought.

For those outside of the maths building, I would like to thank the Southampton University Archery Club for their fun adventures, you will never find a nicer group of people. Matthew, Liam and Patrick, three people I now consider brothers, who kept me sane be that via a few drinks, good food, board games or a combination of all three, thank you so much. Sami, Geoff, Suz, who ever said gingers don't have souls obviously never met these three as they are the most wonderful people with the biggest hearts I've ever seen.

Finally, and most importantly, my Family. Thank you to all of them, and believe me, there are a lot of them, but none deserve it more so than my Mum and Dad, Helen and Trevor. As I sit here now, typing this, I'm struggling to find the right words. In fact, I doubt there are words sufficient to convey everything I want to say. But I'll give it ago. Thank you for never wavering in your support, for giving me the strength to keep going through the tough times, for your patience, understanding and love. Thank you for telling me all my life that I can achieve anything and making me believe it. All of this, and all that is to come, is thanks to you both.

Also my granddad, Michael Cole. One of the best and sharpest men I know, expert Rum Boson and Master of the $\frac{1}{2\pi\sqrt{LC}}$.

"Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world." - A.Einstein

Chapter 1

Introduction

Development of new technologies designed to probe different aspects of biological systems have led to a rapid increase in data generated from these systems [1–5]. For example, these new technologies include next-generation DNA sequencing, microarray technologies and advances in mass spectrophotometry [1–6]. It can be argued that a more complete understanding of how the underlying biological systems work can be gained by combining these heterogeneous data sets and then analysing the results, rather than analysing them separately [7–10]. For example, some features may only become apparent when considered in conjunction with other datasets. Finding clinically meaningful features in patients can then be used for instance for patient stratification, which is crucial in drug trials. The process of combining different data sets is called data integration (DI) [1, 7, 11] and the problem that arises is *how* to combine these data sets in order to provide the most information about the corresponding system.

There are many methods for data integration, all having their strengths and weaknesses, and objective they are based on. Because of this, there is no one method that is perfect for every situation. One method may focus on identifying clusters, another looking for key pathways (a collection of molecules controlling a particular feature) that emerge or looking at how change propagates through the system. Some methods may only be designed for the integration of particular types of datasets, or limited by their computational complexity in the number or size of the datasets. But across these existing methods a range of techniques are used, such as Bayesian methods to estimate the most likely clustering solution, correlation methods to find related features across datasets and network based techniques to find, for example, common clusters. The benefits of DI has been well documented in the literature; for example, iCluster [12] found a previously undiscovered cancer cluster which was not apparent from the separate analysis of the two datasets it used.

It has been commented that more tools utilizing network techniques are still needed [7] and this is where this thesis aims to contribute. Networks can arise quite naturally

from data. Each sample in the given data takes on the form of node, and then the nodes are connected to each other via an edge where the edge captures/encodes the relationship between those two samples (nodes), normally a similarity, or dissimilarity (e.g. distance) measure. For example, if the data contained information about city features the edges could encode how similar each city in the data set is to each other or simply the distance between those cities. The specific metric/measure used is left to the user as some may be more appropriate for one dataset or task than another. Crucially, this network representation often reveals structural information such as clusters, which may not be apparent in the raw dataset.

This thesis' main aim is the development of a new network data integration method to combine multiple networks on the same set of objects but from different sources into one summarising network. The ultimate goal is to identify the global structure in terms of groups of similar nodes, formally clusters in the network. In the biomedical context, where nodes represent patients, this approach increases the predictive power and decision-making capability of the original datasets.

The motivating idea is that we can use a metric, from information theory, to compare the networks seeing how similar/far apart they are in terms of their clustering structure. By using standard minimisation methods, the new method will then return a network that is as close as possible, with respect to this metric, to all of the input networks simultaneously. Afterwards developing our methodology, we apply it to data from cancer patients, in particular using 3 sets of omic data from the patients, and then analyse the results that follow. In the solutions we find we examine the clustering structures and the survival profiles of the respective clusters of patients. This information could then be used to devise better/unique treatments for each cluster or help find new biologically relevant features that identify the cluster.

This thesis is organised as follows. The rest of this Chapter discusses data integration in system biology and network entropy. The first section looks at multiple aspects of data integration methods: the need for it, how they can be categorised, the various stages that they can be performed and existing methods. The second section details the concept of entropy from Information Theory and how it has been used in the analysis of networks. We look at particular methodology, which is not a typical fusion method as a single output network is not guaranteed, that uses entropy in the integration (reduction) of layers in a multilayer network: Structural reducibility of multilayer networks [13]. This introduces a key concept that inspired our own new data integration methodology.

Chapter 2 consists of an in-depth look at a particular integration method, Similarity Network Fusion [14], a competing state-of-the-art data fusion methodology. This method is of particular interest because it is a network based method which utilises the local structure during its integration process. We then examine it from a formal mathematical point of view, as a matrix limit. If the procedure converges to a solution then this can

provide one unquestionable solution and by using mathematical tools this limit can then be calculated, speeding up the method as well. We show that the limit can be analytically computed in the case of two layers, but it is in fact a constant matrix.

In Chapter 3, we initiate our investigation into network entropy and how it can be used in the context of data fusion. We explain how entropy in Information Theory can be adapted to Quantum Mechanics and to Network Theory. The rest of the Chapter explores network entropy: theoretical results on common familes of graphs, examples providing an intuition of what structural properties it captures in networks and how this extends into quantifying the distance between two networks.

Chapter 4 is the main contribution of this thesis and it describes our new data fusion algorithm, Entropy Network Fusion (ENF). First we motivate the method and then study it in more detail. ENF accepts as its input, a family of symmetric matrices, each representing a similarity measure between nodes on one network, and outputs a symmetric similarity matrix representing the 'fused' network. The cost function that ENF is trying to minimise is based upon the average of squared errors using entropy and a quantity called the Quantum Jensen-Shannon divergence and the output is obtained as the solution of a minimisation problem. An adaptation called weighted ENF (wENF) is include for the case when the layers are to be weighted differently. Afterwards, we analytically derive both the first and second derivative of the cost function so that we can use gradient descent to minimise this cost function. Finally we close out the chapter by analysing the algorithm to ensure it behaves correctly. The first derivative is verified by finite difference; we prove that the ENF cost function is a convex problem; the time complexity of ENF, numerically, is shown to be linear with respect to the number of layers and at most cubic with the number of nodes and finally test cases to make sure sensible answers are given.

Chapter 5 consists of two main parts. The first covers the comparison tools that will be used to validate the results from ENF. In the presence of survival data, one such tool is survival analysis. For clusters identified in the data, survival analysis compares the survival curves of each identified group and evaluates whether they are statistically different. We also use network-theoretic measurements to validate the clustering results obtained by ENF, for example by comparing them to the clustering of each individual input network. In the second part of the chapter, we apply ENF to the SNF validation data in [14], along with the results from two naive methods of data integration. The comparison of ENF and SNF is the key focus as we perform a side by side comparison for each of the 5 cancer data sets used in [14]. We find that whilst ENF is computationally slower than SNF, overall ENF out performs SNF. We found evidence that ENF is more robust against noise within the system; that structurally, the ENF solutions are closer to the given input networks and that their clusterings are closer to ENF than SNF, with respect to our validation metrics. With respect to the survival analysis, ENF produced significant results for every dataset and for multiple clusters values.

Chapter 6 focuses on extending the scalability of ENF by reducing its computational complexity. This is done by using a method to approximate network entropy and using this to create a variation of ENF that we call approximate entropy Network Fusion (aENF). This is followed by the derivation of its gradient. We then apply this new variation to two of the data sets already used and then, using the quantities already covered, make a comparison between the two variations. By using this approximation we obtain a 54% and 73% speed up in computational time for the smallest and largest of our test data sets respectively. The outputs from aENF are very close to those from ENF with respect to our various validation measures. This thesis ends with Chapter 7, which summarises and presents conclusions for our work, as well as possible directions for future work.

1.1 Data Integration in System Biology

The development of high-throughput technologies in the Life Sciences, such as next-generation sequencing, microarrays, RNA sequencing or mass spectrometry based proteomics has lead to an overwhelming amount of omic data being produced [1–5, 15]. Omic data refers to multiple collections of genome-scale biological molecules each capturing a different function/aspect of the biological system they originate from; data sets such as genomics, transcriptomics or proteomics [8–10, 16]. For example transcriptomics datasets captures information about RNA transcripts and the active components of the cell; proteomics techniques identify and quantify of proteins within the cell. Together, these data types capture different aspects of the same underlying biological system, mathematically a very complex dynamical system [15, 16]. In this thesis, we will be looking at different omic datasets on the same group of patients with respect to a given disease, in order to combine these multiple datasets into a single one (data fusion).

It has been reported that the pace at which omic data is being produced far surpasses that of Moore's law, which refers to how computing power and storage capacity doubles every 18 months [7, 17]. Scientists use this data to stratify patients into subgroups, relating to medication response or survival prospects, which is used to form treatment decisions [15, 18, 19]. With the availability of this extra data, complex biological systems can, and should, be studied as a whole. This would allow for knowledge discovery that may not be attainable from the study of a single data type [7–9, 15]. Particularly, Liu et al. [10] refer to the *"emergent property"*, relating to how some properties can *"only be observable when the system is studied as a whole and not as the sum of its parts"*. Hence this has led scientists to considering, and performing, data integration (DI).

Among the literature there is some ambiguity behind the term 'Data Integration'. This ambiguity is highlighted in [7] and can be seen in [20] where DI is used in the context of fetching the raw data from multiple sources and bringing it together into one collection. The context that DI is used in this thesis is as described above and in [1, 7, 11]: *'the use of multiple sources of information (or data) to provide a better understanding of a system/situation/association/etc'*. In the context of this thesis, we expect the development of effective DI methods to have a significant impact in system biology and biomedicine.

Let us clarify this ambiguity further with a quick example of a method that will be appearing frequently in this thesis, Similarity Network Fusion (SNF) [14]. Figure 1.1, taken from [14] nicely illustrates SNF, which we shall discuss later in Chapter 2, but more importantly how DI works. Panel (a) of Figure 1.1 starts with two separate datasets, mRNA expression and DNA methylation but the rows in each dataset correspond to the same object. I.e. the information in the first row of each dataset is from the same patient. SNF compares how similar the patients are within each dataset, giving panel (b), and uses networks to visualize this information, giving panel (c). From these two panels we can see the different in structure between the two dataset, even though they

are datasets on the same patients. We assume that each data set captures a partial view of the same underlying biological system. This justifies DI: by combining or integrating the two datasets, we obtain more reliable information than by studying each system separately and then combining the information. In the case of SNF, the networks in panel (c) are combined into the network shown in panel (e), giving one single network output, that has incorporated the information from both datasets revealing details that would not be apparent from either dataset on its own.

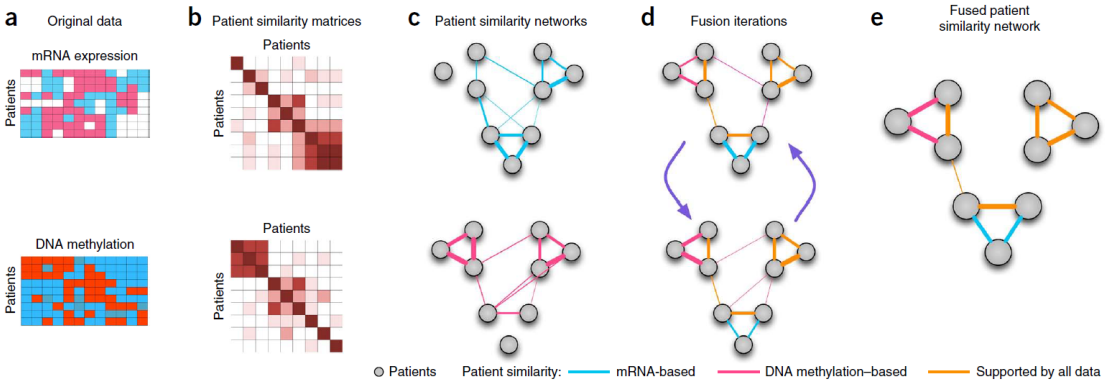

FIGURE 1.1: Illustrating DI with SNF, taken from [14]. (**a**) The raw datasets where the rows in each dataset correspond to the same patients. (**b**) For each dataset a similarity matrix quantifying how similar the patients are to each other within that dataset. (**c**) The similarity matrices are visualized using a network representation. As can be seen there are very different structures even through the data originate from the same set of people. (**d**) The fusion/integration process begins, combining the information from each dataset together into one network. (**e**) The output from the fusion process showing structure and information based on a more complete view of the whole system. Revealing information that would not be apparent from either dataset alone.

Data integration can be done at three stages: an early stage, an intermediate stage or a late stage. An early DI method consists of bringing all the information from the separate sources into one data matrix and then performing an analysis. Intermediate DI involves some initial analysis/representation of the separate data types and then combining those representations. Lastly, late DI involves completely analysing each data type and then combining the results [5, 18, 21]. Figure 1.2 from [11] provides an intuitive illustration of these stages, referred here as concatenation, transformation and model based respectively. These DI methods can then be further categorised in terms of the objective that they seek. In [22] they argue that the objectives of DI methods for omic data can be broadly separated into three categories. The first is the discovery of molecular mechanisms, the second being the clustering of samples, such as patients, and finally the prediction of an event (therapy efficiency, survival etc.).

Initially late DI methods were used, however there has been a shift towards intermediate methods due to a vast amount of studies concluding [7, 10, 22] that intermediate DI methods outperform late DI methods or analyses with no integration at all. This is illustrated particularly well in [12], where the authors present an intermediate DI

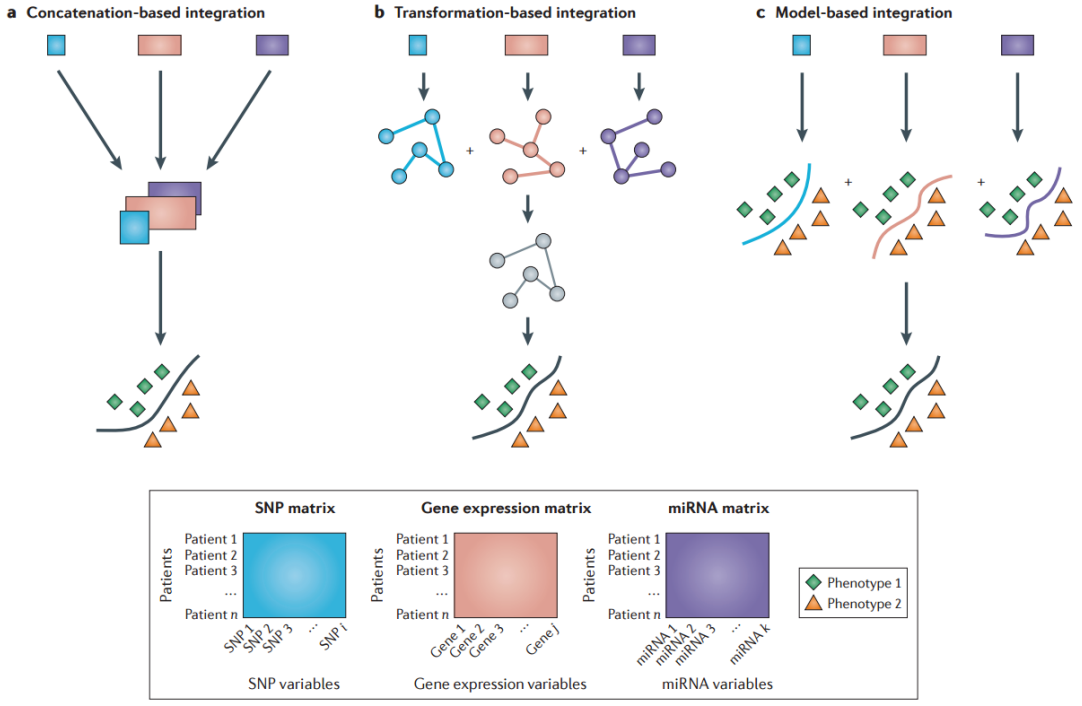

FIGURE 1.2: Illustration of the different stages of DI taken from [11]. (a) Concatenation, or 'early' as we refer to it, has the datasets combined into one set and is then analysed. (b) Transformation, what we call 'intermediate', has each dataset transformed into a different form e.g. a network, which are then combined and then analysed. (c) Model, or 'late' integration, analyses each dataset separately and then the results are combined.

methodology called iCluster. Through their intermediate DI method, Shen et al [12] identified a third, potentially novel, subgroup in the data from [23] which did not appear under a late DI approach. They also identified a fourth cluster, which was not included as it did not show any patterns, though Shen et al [12] remark that patterns might emerge if more data types were to be included (as they were only integrating two datasets). Nevertheless, the discovery of this third subtype illustrates the power of intermediate DI methods.

Many studies have concluded that the development of methods, particularly ones that integrate more than two omic data types, is still needed [7, 10, 11, 15, 22]. Development of DI methods is not an easy task as there are multiple challenges that any DI method has to try and overcome. Challenges include the different dimensionality of each data type, overcoming the noise present in the data, and the effective use of concordant and discordant information. [1, 12, 14, 17]. The contribution of this thesis is precisely the development of a novel intermediate DI method, based on information-theoretic network entropy.

In [22] the authors mention two binary features to describe DI methods, which are: bayesian based and network based. The combination of these methods give rise to

4 categories: network free-bayesian free, network free-bayesian based, network based-bayesian free, network based-bayesian based. A method is said to be bayesian based if the method uses statistics to model the data starting from a reasonable set of assumptions. Similarly, a method is said to be network based if the method uses networks/graphs to represent the data/the interaction between variables. Applications and results from graph theory can then be applied to help draw insight of the underlying biological network. Networks are indeed a popular and widely used tool in the analysis of biological systems [1, 22, 24]. A third binary feature we could include is whether the method is supervised or unsupervised. Supervised methods use prior assumptions, for example, an initial clustering structure or more weights to certain features, or other information external to the data set. Methods without prior assumptions are classed as unsupervised. An important problem with supervised methods is that they can skew the results and fail to detect novel/new information beyond the initial assumptions. The end goal of this thesis is to cluster patients based on the outcome their clinical/biological data without any prior assumptions. In this thesis, we will present a new network based, bayesian free, unsupervised DI method on clinical/biological data based on information theoretic network entropy.

In the public domain there are many intermediate DI methods designed for omic data with the goal of clustering. For example, in the network free-bayesian free category we have Integromics [8] and MCIA [3], which we discuss below. Integromics [8] consists of two approaches for integrating two omics data sets.

The first approach Integromics uses is a regularised version of canonical correlation analysis (CCA) [25,26]. The original CCA [26] seeks out to find linear relationships and correlations between variables from two different datasets on the same set of objects. This is done by finding two vectors, that are a linear combination of the variables from each dataset, that give the largest correlation. Assume we have two matrices X and Y of order $n \times p$ and $n \times q$, where n are the number of samples/objects and p, q are the number of features for each data set. Also both sets of columns are assumed to be full rank, mean centered and have a variance of 1. Let a and b be two vectors of coefficients for a linear combination of the columns of X and Y respectively, which are calculated by Xa and Yb. Then CCA seeks to solve:

$$\rho = \max_{a,b} cor(Xa, Yb).$$

That is, it finds vectors a and b that maximise the correlation between the two linear combinations. Let S_X and S_Y be the sample correlation matrices for X and Y. By forming orthogonal projectors, $P_X = \frac{1}{n} X S_X^{-1} X$ and $P_Y = \frac{1}{n} Y S_Y^{-1} Y$, the solutions to the above maximisation can be found by an eigenvalue decomposition of $P_X P_Y$ and $P_Y P_X$.

This is where the original CCA starts to break when applied in a biological context. A standard condition as to when to apply CCA is when $n > p + q$, that is when the number of samples is greater than the total number of variables. The reason that CCA can not/should not be applied when this condition fails is due to the inverse in the orthogonal projectors. If p (or q) is greater than, or even close to n, then the matrix S_X (or/and S_Y) becomes singular or ill-conditioned. This means that the inverse can either not be calculated, or it is unreliable. This situation is frequent with biological data, where the number of features will often greatly outnumber the number of samples. By introducing a regularisation, the problem can be avoided. The matrices S_X and S_Y are modified by adding on a non-negative multiple of the identity matrix, becoming $S_X + \lambda_1 I_p$ and $S_Y + \lambda_2 I_q$. The rest of the procedure continues as before, with an additional cross-validation step to tune λ_1 and λ_2

The other approach in Integromics, is a sparse version of Partial Least Squares (PLS) [15, 27, 28], which simultaneously performs local regressions on two datasets X and Y, that have the same number of samples (i.e. rows) but different number of columns. It assumes that the matrices can be written in a latent variable model, a technique that we later see with iCluster, which is:

$$X = W_x H_x + \epsilon_x$$

$$Y = W_y H_y + \epsilon_y$$

H_x and H_y are referred to as loading vectors, W_x and W_y are the latent variables (matrices) containing the regression coefficients and ϵ_x and ϵ_y are matrices added to account for the noise/error. The objective of PLS is to find loading vectors that maximise the covariance between W_x and W_y. To make PLS more applicable to biological omic datasets the sparse version (sPLS) was made to help with the computation costs and the interpretations of the results. The solutions are found using the eigenvectors from a singular value decomposition of the product $X'Y$. These are then modified to induce sparsity using a soft-thresholding function.

Multiple co-inertia analysis (MCIA) [3] is an extension of co-inertia analysis (CIA), which was originally only designed for the integration of two dataset on the same set of objects. For the purpose of this explanation we assume that the number of the rows (samples) for the datasets are the same but the number of columns vary. In MCIA it still has the same motivating idea, transforming each dataset to the same (lower) dimensional space an identifying co-relationships between the datasets. Both CIA and MCIA transform the original tables of data into a form that captures how much variance the entries contribute to the data. This new table/matrix X along wuth the matrices D_r and D_c, that describe the amount of variation each row and column contributes, form a triplet which is key for both methods. Both methods then use the triplets for each dataset to form a version of a covariance matrix and then proceed to do an eigenvalue

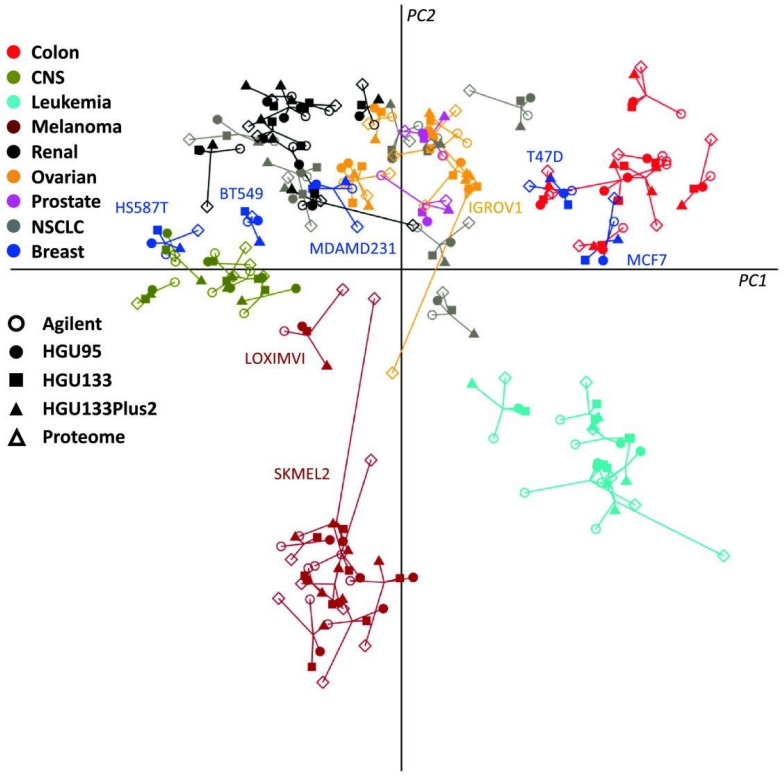

FIGURE 1.3: Taken from [3] this shows a resulting plot from the MCIA method. The colours are used to differentiates the cell lines taken from different tissues and the shapes are used to represent the different datasets.

decomposition. The eigenvectors found from this are then used to create axes for each dataset, all of the same dimension, that maximise the covariance between them. This means that the samples have a set of coordinates for each dataset and can be plotted in the same figure. In this figure, each dataset uses a different symbol for its samples, and then the points representing the same sample in different data sets are join by a line. The length of which is proportional to the divergence between the datasets, the shorter the line the more strongly they agree. In the case of MCIA, as it integrates more than two datasets, it uses an artificial reference centre for each sample from which the covariance is maximised. Therefore in the corresponding plot, the different symbols corresponding to the same sample are connected to the reference centre, as we see in Figure 1.3. The length of the line from the reference centre to the symbol remains proportional to the divergence of that dataset from the reference centre.

Examples in the category of network free-bayesian based are iCluster [12] and PSDF [19]. The main goal of iCluster [12] is to estimate a clustering structure that is common to all of the data sets in a probabilistic framework. This is done by exploiting a connection between K-means, Principal Component Analysis (PCA) and latent variable models. Shen et al. start by discussing how K-means clustering can be solved optimally through PCA. By reformulating the K-means cost function and allowing for a continuous

solution, rather than a discrete one, the problem is then solved by a eigenvalue decomposition of $X'X$, (where the column of X are the data points). Specifically by setting $Z^* = (e_1, \ldots, e_K)'$ where e_i are the eigenvectors associated to the K largest eigenvalues of $X'X$. The discrete structure, Z, can then be restored by either a QR decomposition or a K-means algorithm on Z^*. Now the clustering problem can be represented by a Gaussian latent variable model. That is the data, X, can be written as:

$$X = WZ + \epsilon$$

Here Z is still the cluster matrix as a hidden (latent) variable, W is a coefficient matrix that projects the sample into that space and ϵ is an error term added to account for the noise/unique variances. Under additional assumptions, when considering a continuous solution, Z^*, for this model, a likelihood based solution is available that utilises the solution from above. The uniqueness of iCluster then arises by using this model to estimate a single clustering that is common across multiple data sets, that is:

$$X_i = W_i Z + \epsilon_i$$

This is done by applying a Expectation-Maximization (EM) algorithm to a complete-data log-likelihood function (with a lasso regularization). A likelihood function evaluates how likely the current variable/parameters are given the data that has been observed, the desired outcome is having parameters that maximise this. The EM algorithm does just this by alternating between two steps. First is the E-step calculating the expected solution for Z based on the current parameters and the subsequent step, the M-step, is to update the current W and ϵ to maximise the likelihood for the current solution. These steps are then repeated until the solutions and parameters have converged.

Patient-Specific Data Fusion (PSDF) [19] is a extension of the Transcription Module Discovery (TMD) method in [29] to include feature selection. The method integrates two datasets on a sample by sample basis, the samples being genes, to find samples that are clustered together in both of the two datasets. These genes that are then clustered together are then a potential *transcription module* which are a collection of co-regulated genes that control a common biological function. The model for PSDF and TMD is based on two-level hierarchy of Dirichlet Process Mixture models (DPMs). One level controlling whether the data for a gene should be fused or not, and the second controlling the clustering of the gene depending on if it's fused or not.

Lastly, for network based-bayesian free methods, which are few in comparison to the other categories, we have Multiplex [30] and SNF [14]. Multiplex [30, 31] is not actually a DI method, but is a framework that allows a user to jointly analyse multiple weighted networks on the same set of objects that arise from each of the the different datasets. Each different datasets is capturing a different interaction/aspect of the same system. Given weighted networks G_α on N nodes with adjacency matrix $A^\alpha (= a^\alpha_{ij} \geq 0)$, then

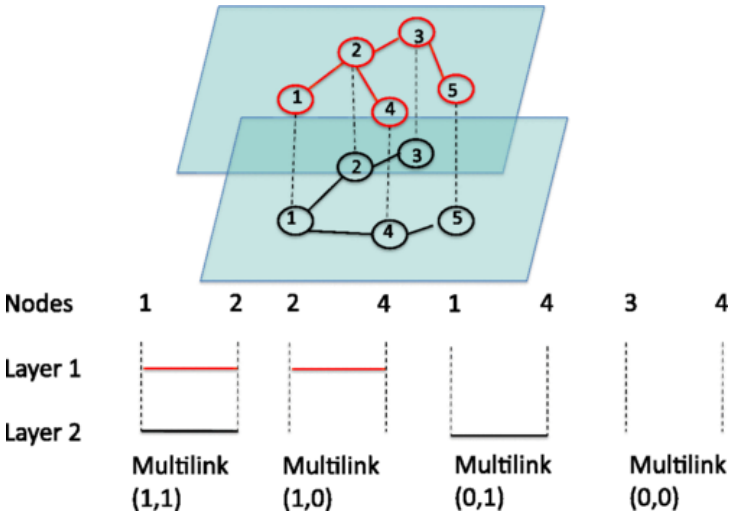

FIGURE 1.4: Here we have an illustration of multilinks in the case of two layers. There are two different networks on the same 5 nodes. A multilink of type $(1,1)$ are pair of nodes $(1,2)$, $(2,3)$ that have edges between them in both layers. Whereas the pair $(2,4)$ would have a multilink of $(1,0)$ as they are only connected by an edge in the first layer. Illustration taken from [30]

the *degree*, *strength* and *inverse participation ratio* for a node i in layer α are defined as:

$$k_i^\alpha = \sum_{j=1}^{N} \tau(a_{ij}^\alpha)$$

$$s_i^\alpha = \sum_{j=1}^{N} a_{ij}^\alpha$$

$$Y_i^\alpha = \sum_{j=1}^{N} \left(\frac{a_{ij}^\alpha}{s_i^\alpha} \right)^2$$

where $\tau(x) = 1$ if $x > 0$ and 0 otherwise. The degree simply captures how many connections a given node has; the strength gives a indication of the importance of the given node and the inverse participation ratio describes how evenly the edge weights of a node are distributed or the number of effective links it has. By analysing these quantities and their correlations information and insight can be gained about its structure. An example is to look at the degree of nodes against their average strength/inverse participation ratio. For a multiplex network, a collection of M weighted networks, these quantities are generalised/extended to incorporate the concept of multilinks. Multilinks, $\overrightarrow{m}$ are vectors of length M that are designed to characterise the multiple links between nodes over the layers. The entries, m_α, are binary indicating if edges exist in layer α, 1, or not 0. For example in the case of two layers the multilink $(1,1)$ would be characterising any pair of nodes that have edges in *both* layers, like nodes 1 and 2 (and 2 and 3) in Figure 1.4. In total there are 2^M different multilinks and a adjacency matrix $A^{\overrightarrow{m}}$ can be made

for each one, according to:

$$A_{ij}^{\vec{m}} = \prod_{\alpha=1}^{M} [\tau(a_{ij}^{\alpha})m_{\alpha} + (1 - \tau(a_{ij}^{\alpha}))(1 - m_{\alpha})]$$

With this the *multidegree*, *multistrength* and *inverse multiparticipation ratio* are defined as:

$$k_i^{\vec{m}} = \sum_{j \neq i}^{N} A_{ij}^{\vec{m}}$$

$$s_{i,\alpha}^{\vec{m}} = \sum_{j \neq i}^{N} a_{ij}^{\alpha} A_{ij}^{\vec{m}}$$

$$Y_{i,\alpha}^{\vec{m}} = \sum_{j \neq i}^{N} \left(\frac{a_{ij}^{\alpha} A_{ij}^{\vec{m}}}{\sum_r^N a_{ir}^{\alpha} A_{ir}^{\vec{m}}} \right)^2$$

Then these quantities can be analysed in the same fashion as the original quantities but now factoring in how these connections are supported across different layers.

Similarity network fusion (SNF) models each data set as a network and takes the local information from each network to propagate them into each others via an iterative process. We will describe this method in more detail in Chapter 2, as we use it for comparison and validation with our own novel DI method, Entropy Network Fusion (ENF).

We have discussed several DI method, all of which have their own motivations with resulting advantages and disadvantages. For example, all the methods in Intregromics (CCA,RCCA,PLS,sPLS), CIA and PSDF only allow for the integration of 2 datasets, arising from the design of the method. If a more complete understanding of a system is to be had, then a DI method will need to be able to fuse more layers. Multiplex is a framework for analysis of multiple networks rather than a DI method, analysing the connections between nodes over the different layers. Because of its formulation and complexity there is no apparent consensus on how to do clustering in a multiplex. In MCIA, the raw datasets are transformed into one summarising the variation in that dataset, this is repeated for all of the input datasets regardless of their origin. In some cases this could be a too simplistic or an inappropriate treatment of the data. Some datatypes might need to be treated differently or relationships between data might not appear with such a simple treatment. Ideally a DI method should be able to handle data from different sources appropriately. As for iCluster there is a potential issue with computation complexity and prior assumptions. The calculations used in iCluster become complex as the number of features rise, so a pre-selection of the features (genes) to be used is typically done, using prior knowledge. This introduces the added risk of omitting data that could be useful under the joint analysis or introducing a form of bias. Note that in Wang's paper [14] where they introduce SNF, the method we shall

be comparing ours to, they did a survival analysis (see Section 5.1.1) using SNF and iCluster and compared the results. In it SNF out performed iCluster and they found that iCluster was extremely sensitive with respect to the pre-selection of genes. Therefore if our method out performs SNF, that implies it beats iCluster as well.

With our method we shall avoid having these issues. Our method will not require a pre-selection of features, we shall be using a network representation encoding the similarities between samples using all of the features if desired. The actual construction of the networks is left to the user, allowing for different similarity measures to be used for each dataset/data type. This and the ability to fuse more than 2 datasets, will give our method a level of generality such that it will have a wider range of applications and not be restricted to biological datasets only. Further by using a network representation we shall also fulfill the demand for more network based DI methods, whilst being able to draw on a whole area of results. For example, we've seen eigenvalues used in some methods but not used in any using network methods. Eigenvalues of networks have a connection to its clustering structure, in our own method we shall exploit this, as our end goal is to cluster the nodes. Information-theoretic entropy, uses the eigenvalues of a network to analyse it, this we discuss in the next section.

1.2 Network Entropy

Information-theoretic entropy has begun to emerge as a network analytic tool, which we will discuss in this section. Shannon's seminal paper, *A mathematical theory of communication* [32], established the formulation of information-theoretic entropy and has since expanded into a vast area [33]. In his paper, Shannon considers the problem of measuring the uncertainty of a given event. Suppose that we have n events, with probabilities $p_1, \ldots, p_n$. Shannon considers three properties that the sought-after uncertainty measure should have. The first is that it should be continuous in the probabilities p_i. The second is that, for uniform probabilities, $p_i = \frac{1}{n}$ for all i, the measure should be a monotonic increasing function of n. (That is, if all events are equally likely, the uncertainty should increase as the number of events increase.) Finally, if an event can be broken into two successive events then the total uncertainty should be the weighted sum of uncertainty of the two sets of events. Shannon then proves that the only function to satisfy these properties is:

$$H_{\mathcal{C}} = -K \sum_{i=1}^{n} p_i \log_2 p_i, \tag{1.1}$$

for K a arbitrary positive constant. Note that this quantity $H_{\mathcal{C}}$ is similar to the entropy that appears in statistical mechanics [34], and hence the name. Also note that the base of the logarithm is not restricted to be 2 as it corresponds to a choice of units, 2 is selected so that the units. (In this case, the base 2 logarithm means the units of information are bits.)

This quantity has been applied to analysis of classical networks [35]; used in cluster comparison measures [36]; in measures for knowledge discovery and data mining [37]. Gradually it has been applied to complex networks. When applied to complex networks it can be interpreted as quantifying how structured the network is based upon its key features or, alternatively, the complexity of the system [38–43]. For example, in [44] the authors quantify the dissimilarity between networks by using an entropy measure with the degree distribution of the nodes. When clustering a network in [45] the authors use entropy to evaluate the stability of the clusters and find any unstable nodes the network may contain. Entropy is used in [46] to measure local randomness of integrated protein interaction and mRNA expression networks to identify genes and pathways indicating a more aggressive phenotype. More recently, entropy has been used in the data visualisation method t-SNE, which reduces high dimensional points to a lower dimensional space whilst preserving as much of the structure as possible [47]. In [13], which shall be discussed in more depth in the next section, entropy is used to reduce redundancy that can appear in multilayer networks. This last technique has been applied in the analysis of brain activity [48, 49], with much success. The novelty of this thesis is the use of network entropy as motivation for a DI method, as we will explain in Chapter 4.

1.3 Structural Reducibility

This section discusses in more detail the techniques used by De Domenico et al in their article *Structural reducibility of multilayer networks* [13]. In this article they use entropy as a measure of how much redundant information two networks on the same set of vertices share. Then, they develop a data integration methodology to reduce the number of layers minimising the information loss.

1.3.1 Summary and Results

A multilayer network (MN), $\mathcal{M}$, is a collection of networks on the same set of n vertices $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each of these networks, referred to as a layer, represents a different feature/relationship between these vertices. For example, the edges of a network could be time dependent thus the layers correspond to the network at different points in time. If these layers have edges between them, such as edges identifying nodes in different layers, this construction is known as a multiplex [13, 41, 48, 50]. These MN can become very large in terms of number of vertices and layers, with potentially lots of unique and redundant information. Therefore a methodology to reduce this redundancy could lead to a more compressed and effective representation of the MN, reduce the complexity when performing basic structural analysis and even lead to insights about the source of the layers and the relationship between them. This is the motivation behind the methodology introduced by the authors, which we describe next.

Network	N	M	M_{opt}	max$[q(\cdot)]$	χ
Human HIV-1	1006	5	2	0.499	0.75
Terrorist network	78	4	2	0.239	0.67
Candida	368	7	4	0.527	0.50
Plasmodium	1204	3	2	0.500	0.50
S. cerevisiae	6571	7	4	0.115	0.50
Xenopus	462	5	3	0.424	0.50
FAO Trade Network	184	340	182	0.354	0.47

TABLE 1.1: A sample of results from application of the given method, more can be found in [13]. N denotes the number of vertices in the network. M the number of layers in the original system. M_{opt} the number of layers in the reduced system and the distinguishability it achieves, max$[q(\cdot)]$. χ corresponds to the reducibility of the system.

De Domenico et al's method proceeds as follows (see the next section for details): First, using entropy, the Quantum Jensen-Shannon distance between all the pairs of layers from the MN is computed. Then, the unique pair that realises the smallest distance is then selected to be merged (integrated), which in this case is simply done by summing the adjacency matrices, giving a reduced MN. This process then iterates until only one network is left, the aggregated graph. The order at which these layers are combined can be viewed as a hierarchical clustering of the layers. However, at each iteration, the distinguishability from the aggregated graph is calculated and the reduced MN that maximises the distinguishability is given as the result. In an ideal situation, rather than selecting the pair that omits the smallest distance at each stage, you would consider all possible partitions of the layers instead. However, the complexity of this would make it computationally expensive, therefore pairs are only considered as a compromise.

De Domenico et al apply their methods to a vast selection of MNs, such as the gene interactions in 13 different organisms from BioGRID [51], co-authorship networks, social systems and continental air transportation systems. The variety of these MN also highlighted the adaptability of the method, with vertices ranging from 78 to 14065 and layers from as few as 3 up to 340. Table 1.1 contains a sample of results, and the complete set can be found in the original paper. As an example of the power of their data integration entropy method, consider their analysis of the FAO (Food and Agriculture Organisation of the United Nations) network. Each layer represents a product, with nodes representing countries and edges the import/export of the product between those countries. With their method, the early stages of the aggregation corresponds to products that have similar import export patterns, hence information is gained about the grouping of products and the countries they travel from/to. This led to a reduction from 340 layers to 182, close to 50%.

However, whilst this may seem like effective method it is not quite a proper/full DI method. In the context that we are considering a DI give one single result in its output. Whereas with this method we see that this is clearly not the case, as illustrated in Table

1.1, infact in the following breakdown of the method we see that it actively tries to avoid giving one output. This still leaves the user with the issue of how to reconcile the multiple ouputs and how to proceed forward. The method we will develop avoids this issue by having a cost function to find *one* network that is as close as possible to all the given inputs.

1.3.2 Entropy and Multilayer Networks

Whilst this subject forms the main content of this project and will be studied in depth in Chapter 3, we quickly give the necessary definition to complete the review of [13]. To calculate the entropy of a network given its adjacency matrix G, the first step is to form the rescaled Laplacian matrix, sometimes called the *density matrix* [35], $\bar{L}(G)$, where the eigenvalues sum to 1.

$$\bar{L}(G) = \frac{L(G)}{Tr(L(G))} = \frac{D_G - G}{\sum_{ij} G_{ij}}$$

Where D_G is the degree matrix of the graph G. Then the entropy is given by:

$$H(\bar{L}(G)) = -\sum_{\lambda_k \in \sigma(\bar{L}(G))} \lambda_k \log_2(\lambda_k), \tag{1.2}$$

where λ_k are the eigenvalues of the rescaled laplacian matrix , and if any of the eigenvalues are equal to zero we set $0 \log_2(0) = 0$. Using this, the distance between two networks can be quantified using the Quantum Jensen-Shannon distance [43, 52] and this is the metric used to select the two layers that are to be merged. Given two networks, G_i and G_j, their distance is given by:

$$J(G_i, G_j) = H\left(\frac{\bar{L}(G_i) + \bar{L}(G_j)}{2}\right) - \frac{H(\bar{L}(G_i)) + H(\bar{L}(G_j))}{2}. \tag{1.3}$$

The unique pair $(i \neq j)$ of networks that achieves the smallest value in this metric are then selected to be fused together, by adding their respective adjacency matrices together. The reasoning is that it is more desirable to fuse together layers that are the most similar in structure (least information loss).

Once two layers are fused, they are removed from the MN and the new fused layer, the summation of the two, is added in their place giving a new, reduced, multilayer network $\mathcal{R} = \{R_1, \ldots, R_l\}$. The average entropy of this reduced MN is then calculated, given by:

$$\bar{H}(\mathcal{R}) = \frac{H(\mathcal{R})}{l} = \frac{\sum_{R_i \in \mathcal{R}} H(\bar{L}(R_i))}{l}. \tag{1.4}$$

Where l is the number of layers in the reduced MN. The next step is to compare the reduced MN to the aggregated graph from the original MN by using the average entropy. Here, $\mathcal{A}$, is used to denote the aggregated graph from the original MN and $H_{\mathcal{A}}$ is used to denote its entropy. The reduced MN is then evaluated by using the *distinguishability* of $\mathcal{R}$ from $\mathcal{A}$, which is given by:

$$q_A(\mathcal{R}) = 1 - \frac{\bar{H}(\mathcal{R})}{H_{\mathcal{A}}}, \tag{1.5}$$

Once calculated, the distinguishability and the reduced network is stored, and the next iteration begins. This process is then repeated until the reduced MN, $\mathcal{R}$, has been reduced to one layer, the aggregated graph. Upon completion the iteration that yielded the largest distinguishability is selected and given as the solution.

The motivation behind this is that the best reduction possible is combining all the layers into one, forming the aggregated graph. However in some situations doing this can result in key structural information being lost. Information that appear in some layers but not in others which then become lost in the cumulative noise. Hence by seeking the reduced MN where the distinguishability is maximal, this will result in the fewest layers with the most information possible. (One could think of it as the most 'Dense' reduced MN), in terms of information.

The *reducibility* of the MN, $\mathcal{M}$, is then defined as:

$$\chi(\mathcal{M}) = \frac{M - M_{opt}}{M - 1}, \tag{1.6}$$

where M is the number of layers in the initial MN and M_{opt} is the number of layers in the solution. This quantity is the ratio between the number of reduction made, $(M - M_{opt})$, and the maximum number of reductions possible, $(M - 1)$. Figure 1.5 is an image illustrating the procedure, taken from the original paper.

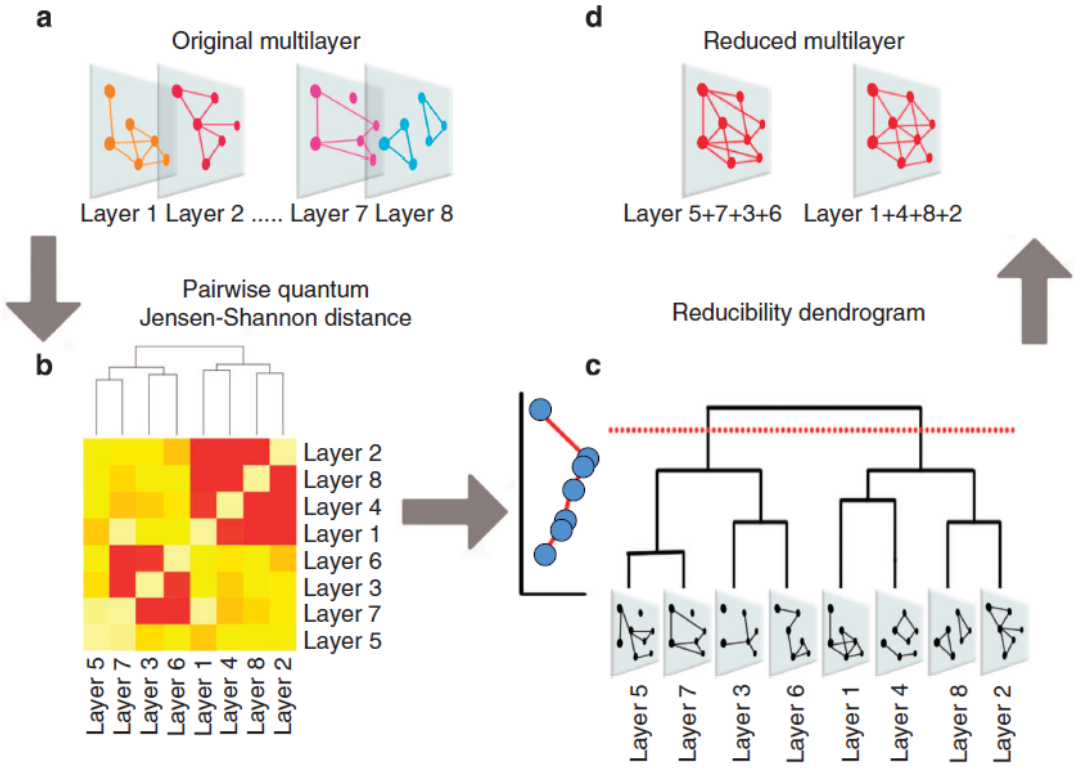

FIGURE 1.5: Illustration of reduction of a multilayer network using entropy. Taken from [13]. (**a**) Initial MN given as an input. (**b**) For all pairs of layers the Jensen-Shannon distance is calculated. The pair achieving the smallest distance is selected to be fused by adding together their respective adjacency matrices, giving a reduced MN. (**c**) This process is then repeated with the reduced MN until it becomes one matrix, forming a hierarchical clustering. At each each stage the distinguishability from the aggregated graph is recorded. (**d**) The clustering maximising the distinguishability is given as the output.

Chapter 2

Similarity Network Fusion

This Chapter will focus on a more in depth look at *Similarity network fusion for aggregating data types on a genomic scale* by Wang et al. 2014. [14]. Similarity network fusion (SNF) is a network based-bayesian free DI method that integrates patient specific data that aims to cluster the patients into potentially novel groupings with different survival profiles. Because of this it is a populate state-of-the-art method and falls into the same category where we shall contribute, it also forms the start point of thesis' work, hence why we shall be taking a deeper look at it. This Chapter is split into two parts, first is a review of the original method, second is original work done at the start of this thesis trying to improve SNF by seeing if a limit can be taken.

2.1 Review

2.1.1 Summary and Results

The authors apply their new method to five different cancer data sets (lung, colon, glio, breast and kidney), integrating data from three different sources (mRNA expression, DNA methylation and miRNA expression). The size of these data sets, in terms of patients, range from 92 (colon) to 215 (glio). Having applied SNF, the result is then clustered using spectral clustering [53, 54]. These clusters are then evaluated using two metrics: a cluster analysis using a silhouette score [55] and a survival analysis using a log-rank test and calculating the associated p-values [56]. The p-values obtained from their analysis is included in Table 2.1 for completeness.

The main steps, discussed in more detail below, of the SNF methodology are as follows. First the data is preprocessed, removing any outliers, filling in any missing data and then normalising the features. Next, for each data type, if not supplied, a similarity network is generated. In the package supplied by Wang et al this is done by using the

Cancer Type	Number of cluster	p-value
GLIO	3	0.0002
Breast	5	0.0011
Kidney	3	0.029
Lung	4	0.02
Colon	3	0.00088

TABLE 2.1: Results of SNF applied to their 5 cancer sets [14], including the number of clusters and the p-values obtained from a log-rank test.

Euclidean distance between the patients (the rows of the data type). From this a full and sparse kernel is generated which captures the full and local similarity of that data type. The integration step then begins, which entails using matrix products with the various sparse kernels to update the full kernel. This is an iterative process and once the desired number of iterations is complete the output is given by an average of all the full kernels. Next we describe these steps in detail.

2.1.2 Pre-Processing

Before the method is applied the data must be preprocessed ('cleaned' in data analysis terms) before it is in a suitable format for the method to be applied. Wang et al performed three pre-processing steps to their data sets.

Firstly the authors remove any outliers that the data may contain. Secondly they filled in any missing data. If a patient or a biological feature had 20% or more missing data then this patient/feature was removed. If however, the missing data for this patient/feature was below this threshold then it was filled in using K nearest neighbour (KNN) imputation [57]. The last step in pre-processing was to normalise each feature of the data. Once these three steps have been completed then the data is considered ready to have SNF applied to it.

The inputs have two components for each data type, a data matrix and a similarity network. The data matrix, M, is a matrix containing the raw data, which has n rows, the patients, and m columns, the measurements for each patient. Each row of this matrix corresponds to all of the information for a single patient. The other input is a similarity network on the set of patients to indicate which patients should be considered as similar to each other. The formulation of this is left to the user, for example K nearest neighbours or an all-to-all euclidean distance matrix. These similarity network are denoted by $G^{(k)}$, the superscript $^{(k)}$ referring to the k-th data type or source.

2.1.3 Local and Global Networks

A weight matrix is used to capture all the patient similarity information. This is done for each layer independently. The weight matrix is formed from the input graph $G^{(k)}$ by the following:

$$W_{ij}^{(k)} = \exp\left(-\frac{\rho^2(x_i, x_j)}{\mu\epsilon_{i,j}^{(k)}}\right). \tag{2.1}$$

The function $\rho(x_i, x_j)$ is the euclidean distance between patient i and patient j from the corresponding data matrix. The term $\epsilon_{i,j}^{(k)}$ is defined as:

$$\epsilon_{i,j}^{(k)} = \frac{mean(\rho(x_i, N_i^{(k)})) + mean(\rho(x_j, N_j^{(k)})) + \rho(x_i, x_j)}{3}. \tag{2.2}$$

Here, $N_i^{(k)}$ stands for the neighbours of the vertex x_i in the graph $G^{(k)}$, thus this is an average of the average distance from x_i and x_j to their respective neighbours and their distance to each other. The parameter μ is used to control the scaling, recommended by the authors to be set in the range of $[0.3, 0.8]$. Note that $1 \geq W_{ij}^{(k)} > 0$. The more dissimilar two patients are, the closer to zero the corresponding entry on the matrix is. If two patients have identical data points then the corresponding entry will be equal to one. In particular the diagonal entries of the matrix are all equal to one.

From this the so-called *full kernel*, denoted by $P^{(k)}$, is formed. This is done via a normalisation of the weight matrix above:

$$P_{ij}^{(k)} = \begin{cases} \frac{W_{ij}^{(k)}}{2\sum_{l\neq i} W_{il}^{(k)}} & \text{if } i \neq j. \\ \frac{1}{2} & \text{if } i = j. \end{cases} \tag{2.3}$$

This normalisation makes it so that the rows of the matrix sum to one but such that the diagonal entry is equal to a half. Next we form the sparse kernel, $S^{(k)}$, which is produced in a similar fashion except in a given row only the information from the k nearest neighbours is used. Hence this is only taking the local information into account. This sparse kernel is formulated as follows:

$$S_{ij}^{(k)} = \begin{cases} \frac{W_{ij}^{(k)}}{\sum_{l\in N_i'^{(k)}} W_{il}^{(k)}} & j \in N_i'^{(k)}. \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

Here $N_i'^{(k)}$ are K nearest neighbours (KNN) of patient i in $G^{(k)}$ for a given K, this includes themselves. This is equivalent to setting the values of points non-neighbouring to zero. The underlying motivation behind this is that the information from local similarities are more likely to be reliable than those further away. This completes the components needed for the algorithm.

2.1.4 Fusion Step

Last is the integration process itself. The motivating idea is to infuse the full kernels with the full information from each of the other full kernels. This will subsequently reinforce or weaken the local similarities depending on how that similarity is supported across the other data types. These kernels are updated simultaneously by taking an average of the multiplication of the other full kernel with the corresponding sparse kernel. This is an iterative process where the number of iterations is chosen before the process begins. The process is defined below:

$$P_{t+1}^{(k)} = S^{(k)} \left(\frac{\sum_{l \neq k} P_t^{(l)}}{m-1} \right) (S^{(k)})^T. \tag{2.5}$$

For clarification $P_{t=0}^{(k)} = P^{(k)}$ and m is the total data types. After each iteration the full kernels are normalised again, rows summing to one and a half on the diagonal, as in Eq. 2.3. This is to ensure that each patient is most similar to themselves and that the final network is full rank. Wang et al report that doing this also leads to a quicker convergence, needing about 20 iterations to converge. The method is defined to have converged if the largest difference between iterations is smaller than 10^{-6}. Lastly, the output is given by the average of the $P_t^{(k)}$ at the final iteration t:

$$F = \frac{\sum_k P_t^{(k)}}{m}. \tag{2.6}$$

Figure 2.1 is an image taken from [14], to illustrate the integration process.

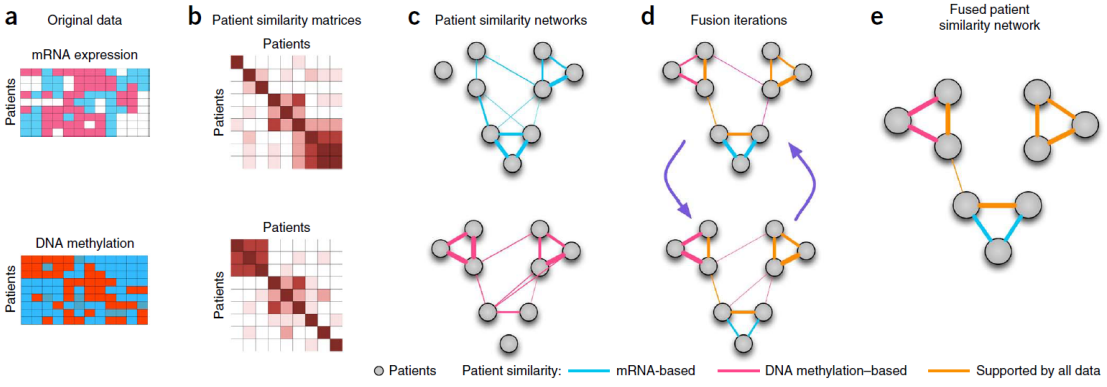

FIGURE 2.1: Illustration of the SNF algorithm taken from [14]. (**a**) The raw data from each source where the rows correspond to the patients. (**b**) For each data type a similarity matrix quantifying how similar the patients are to each other is constructed. (**c**) The similarity matrices are translated into similarity networks. (**d**) The fusion process begins, updating each layer with the information from the other layers. (**e**) An average of all the layers is given as the final output.

2.2 The limit of SNF

As SNF is an iterative algorithm, the process stops when it has 'converged' numerically, that is, the difference between successive iterations becomes very small, less than 10^{-6} in the implementation by the authors [14]. It is therefore natural to ask whether a limit of this process exists. If it does exist, the iterative procedure could be replaced by the limit matrix, and the general conditions for the limit to exist can be studied. This would shed light into the algorithm itself and its general applicability in a wider network data integration setting.

2.2.1 Two Layers

For us to be able to calculate the limit, the process first required two main changes. We first removed the normalisation step, to allow the SNF process to be written as a recursive relation, without affecting the limit. One of the reasons for the normalisation in the first place was to accelerate the convergence, therefore its removal should have no effect other than slowing down the convergence. Also with calculating the limit this becomes redundant. Another reason for its removal was because the operation could not be achieved with linear matrix operations, meaning that the process could not be written as recursive relations. The second main change was to ensure that each of the $S^{(k)}$ are strongly connected and to scale it by $\frac{1}{\rho(S^{(k)})}$, where $\rho(S^{(k)})$ is the spectral radius of $S^{(k)}$. These are conditions required to apply results for computing the limit (see Appendix B). To ensure the $S^{(k)}$ are strongly connected, two solutions were available. The formulation of $S^{(k)}$ comes from using KNN, the first solution is to select enough neighbours so that the resulting matrix is strongly connected. The other solution would be to make the $S^{(k)}$ symmetric, as the KNN algorthim is not a symmetric relationship. Then it would be enough to make sure that enough neighbours are selected to make the matrix connected, as symmetric connected implies strongly connected. Due to the antisymmetric nature of KNN it may capture information that would be lost upon making it symmetric, therefore we chose the former solution.

To begin with, we considered the simple case of two layers. From the definition for the iterative process (Eq 2.5) there are two sets of equations.

$$P^{(1)}_{t+1} = S^{(1)} P^{(2)}_t (S^{(1)})^T. \tag{2.7}$$

$$P^{(2)}_{t+1} = S^{(2)} P^{(1)}_t (S^{(2)})^T. \tag{2.8}$$

Substituting one into the other leads to the recursive relations:

$$P^{(1)}_{t+2} = S^{(1)} S^{(2)} P^{(1)}_t (S^{(1)} S^{(2)})^T. \tag{2.9}$$

$$P^{(2)}_{t+2} = S^{(2)} S^{(1)} P^{(2)}_t (S^{(2)} S^{(1)})^T. \tag{2.10}$$

Solving recursively leads to:

$$P_{2t}^{(1)} = (S^{(1)}S^{(2)})^t P_0^{(1)}((S^{(1)}S^{(2)})^t)^T. \tag{2.11}$$
$$P_{2t}^{(2)} = (S^{(2)}S^{(1)})^t P_0^{(2)}((S^{(2)}S^{(1)})^t)^T. \tag{2.12}$$

At this stage the process is reduced to raising two matrices to a given power t and this is where the limit can be taken. This gives:

$$P_{\infty}^{(1)} = L_{12}P_0^{(1)}(L_{12})^T, \tag{2.13}$$
$$P_{\infty}^{(2)} = L_{21}P_0^{(2)}(L_{21})^T, \tag{2.14}$$

where

$$L_{12} = \lim_{t\to\infty}(S^{(1)}S^{(2)})^t,$$
$$L_{21} = \lim_{t\to\infty}(S^{(2)}S^{(1)})^t.$$

So that the output of the process, on two layers, is given by:

$$F = \frac{1}{2}\left[L_{12}P_0^{(1)}(L_{12})^T + L_{21}P_0^{(2)}(L_{21})^T\right]. \tag{2.15}$$

This formulation reduces the process to calculating the limits L_{12} and L_{21}, which from the results in Appendix B, reduces further to calculating a left and right eigenvector of the two products, $S^{(1)}S^{(2)}$ and $S^{(2)}S^{(1)}$. In essence the limit has all but one of the eigenvalues tend to zero and therefore the eigenvector corresponding to the remaining eigenvalue dominates the matrix powers. Note that the right or left principal eigenvectors of a matrix correspond to a natural measure of centrality [58, 59]. Therefore, for the two-layer case, we have a complete answer: a limit matrix which depends on the initial full kernel in each layer, and the eigenvector centrality of the products of the sparse kernels.

With this result the next step was to generalise this result to three or more layers. However this small change makes the problem significantly harder as we can not simplify the products as before. In the three layer case, proceeding in the same way as in the two layer case, the equations from the iterative process have the form:

$$P_{t+1}^{(i)} = S^{(i)}P_t^{(j)}(S^{(i)})^T + S^{(i)}P_t^{(k)}(S^{(i)})^T, \tag{2.16}$$

which is in terms of the other two matrices, meaning that it is not possible to eliminate both matrices and write the update equations in terms of the matrix we are updating.

Consider one of these equations after just one step and how its complexity has increased:

$$P_{t+2}^{(1)} = S^{(1)} \left(S^{(2)} P_t^{(1)} (S^{(2)})^T + S^{(2)} P_t^{(3)} (S^{(2)})^T \right) (S^{(1)})^T + S^{(1)} \left(S^{(3)} P_t^{(1)} (S^{(3)})^T + S^{(3)} P_t^{(2)} (S^{(3)})^T \right) (S^{(1)})^T. \tag{2.17}$$

On l layers and after t steps in time, the equations would have $(l-1)^t$ terms. This complexity means that if the calculation of the limit was possible, other than by numerical means, it would be extremely difficult. In any case, we found that the limit actually tends to a constant matrix which we then subsequently proved, (shown in the next subsection), and a constant solution is of no practical use. Therefore it shows that SNF is highly dependant on the number of iterations as to the quality of the solution.

2.2.2 SNF Tends to a Constant Matrix

Initially, to show that the limit of SNF is a constant matrix, the maximum absolute difference between the limit and the original method was analysed as the number of iterations was increased. The results can be seen in Figure 2.2, the main plot shows the difference at each stage up to $10,000$ iterations on a log scale. In the top right corner we also include a plot of the first 50 iterations on a linear scale. As can be seen, the initial difference decreases sharply before quickly slowing down. Given that all the entries are bounded between zero and one the size of this change is still significant, but does gradually decrease. This does imply that given enough iterations the difference would tend to zero, supporting the conclusion. Intuitively, the information eventually 'diffuses' through the rest of the layers completely, giving a uniform output.

Now we prove that this is the case: in the limiting case, SNF converges to a constant matrix. The normalisation in the formulation of the sparse kernel (Eq 2.4) makes all the rows sum to one, that is:

$$S^{(k)} \mathbb{1} = 1 \cdot \mathbb{1}, \tag{2.18}$$

where $\mathbb{1}$ is the column of all ones. Having being given this vector we can instantly apply Theorem 8.3.9 from [60] to deduce the spectral radius. The theorem says that there are no nonnegative eigenvectors of $S^{(k)}$ except for multiples of the Perron eigenvector p, the vector associated with the spectral radius. This means that $v = \beta^{-1}\mathbb{1}, \beta \in \mathbb{R}$ and:

$$S^{(k)} v = 1 \cdot v, \tag{2.19}$$

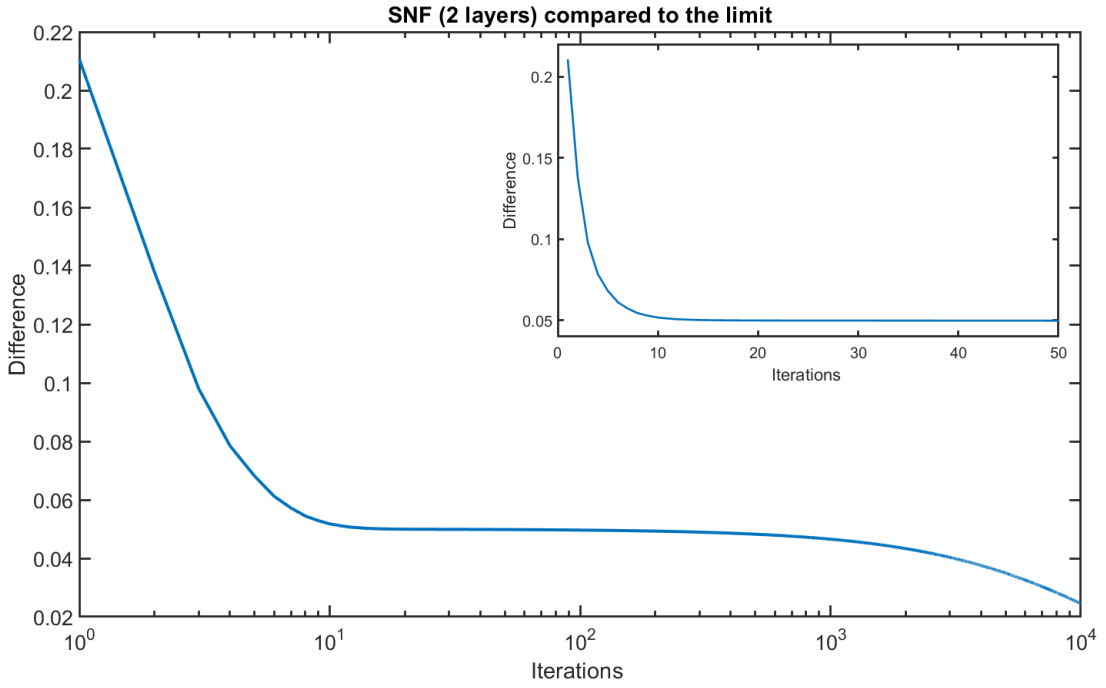

FIGURE 2.2: Here we plot the maximum absolute value of the difference between the limit of SNF and the original SNF method as the number of iterations increase, given on a log scale. The plot in the top right corner shows the first 50 iterations on a linear scale.

meaning that $\rho(S^{(k)}) = 1$. Note here that all of the entries of the Perron eigenvector are the same. The product of two sparse kernels also have rows that sum to one.

$$\begin{aligned}(S^{(j)}S^{(k)})\mathbb{1} &= S^{(j)}(S^{(k)}\mathbb{1}) \\ &= S^{(j)}\mathbb{1} \\ &= \mathbb{1} \end{aligned} \tag{2.20}$$

Hence the product also has spectral radius equal to one and a right Perron vector where the entries are the same. Now consider the limits formed in Eq. 2.13, which are formed from the product of the left and right Perron vector (Eq B.12). Let x denote the right Perron eigenvector, remembering $x_{i1} = x_{j1} = \alpha, \alpha \in \mathbb{R}$ as shown above, and let y denote the corresponding left Perron vector. When computing the limit this gives:

$$\begin{aligned} L_{ij} &= \frac{1}{yx}(xy)_{ij} \\ &= \frac{1}{\alpha \sum_{l=1}^{n} y_{1l}} x_{i1} y_{1j} \\ &= \frac{1}{\alpha \sum_{l=1}^{n} y_{1l}} x_{k1} y_{1j} \quad \text{as } x_{i1} = x_{k1} \\ &= L_{kj}, \end{aligned} \tag{2.21}$$

a $n \times n$ matrix where all of the rows are equal. If a matrix is of this form it shall referred to as *equal-row equal*. When a row equal matrix is multiplied with any other matrix:

$$\begin{aligned} (LP)_{ij} &= \sum_{k=1}^{n} L_{ik} P_{kj} \\ &= \sum_{k=1}^{n} L_{lk} P_{kj} \quad \text{from Eq 2.21} \\ &= (LP)_{lj}, \end{aligned} \tag{2.22}$$

this results in another equal-row matrix. Finally, consider the product LP^T where both L and P are equal-row matrices. Then:

$$\begin{aligned} (LP^T)_{ij} &= \sum_{k=1}^{n} L_{ik} (P^T)_{kj} \\ &= \sum_{k=1}^{n} L_{ik} P_{jk} \\ &= \sum_{k=1}^{n} L_{ak} P_{bk} \quad \text{as } L \text{ and } P \text{ are row equal} \\ &= (LP^T)_{ab}. \end{aligned} \tag{2.23}$$

This is a matrix where every entry is equal. Therefore a product of the form LPL^T like that we had in our limit method, Eq. 2.13, results in a constant matrix.

Next we move towards forming our own data integration method. This method will draw upon the knowledge and insight gained from the study of the papers in Section 2.1 and Section 1.3. It will use entropy to capture the structural information from each layer and use it to fuse the networks in a regression style fashion to find the one network that is most similar to all of the given input networks. First we look at entropy and its definition on networks in more detail.

Chapter 3

Network Entropy

In this thesis we use information-theoretic entropy as a cost function for our fusion algorithm (ENF). Particularly, we shall be using the Jensen-Shannon Divergence as a measure of distance between networks. In this chapter, we first discuss entropy, making the transition from the classic definition, through to the quantum definition and then, its adaptation to networks. After this, we present some formal results for network entropy on some common families of graphs. Section 3.3 then discusses and shows examples to help understand what entropy is capturing in networks. The final section discusses the Jensen-Shannon divergence, which is defined in term of the network entropy, and forms the basis of our data integration cost function.

3.1 Classic to Quantum to Network

As discussed in Section 1.2, Shannon's formulation [32] of entropy is used to quantify how much uncertainity there is in a given (discrete) probability distribution $P = [p_1, \ldots, p_n]$,

$$H_{\mathcal{C}}(P) = -K \sum_{i=1}^{n} p_i \log_2 p_i,$$

with the convention $0 \log_2(0) = 0$. Where K is a positive constant to facilitate a change of base for the logarithim. However entropy is considered in terms of Bits, base 2, therefore proceeding forward K is set equal to 1. To compare two distributions, we can use the *Kullback-Liebler Divergence* or *Relative Entropy* [33, 61–63].

$$KL_{\mathcal{C}}(P|Q) = \sum_{i} p_i \log_2 \left(\frac{p_i}{q_i} \right), \tag{3.1}$$

with the conventions $0 \log_2(\frac{0}{0}) = 0, 0 \log_2(\frac{0}{p_i}) = 0$ and $p_i \log_2(\frac{p_i}{0}) = \infty$, when $p_i \neq 0$. It is also non-negative and only equal to zero when $P = Q$ [33, 61]. This quantifies how much

the distribution P diverges from Q. For example, consider two dice and suppose we do not know whether they are fair or biased and we wish to select the one that is closest to being fair. After 100 rolls and recording the outcomes, the resulting observed probability distribution shows die 1 only rolls 1's, 2's and 3's with equal probability whilst die 2 only rolls 6's. Comparing these distributions to that of a fair die we get a KL divergence of 1 and 2.585 for each dice respectively. This shows that, out of the two available options, die 1 diverges least from that of a fair die and the one that should be selected. The Kullback-Liebler divergence does come with a couple of caveats, namely, that it is unbounded and not symmetric. The anti-symmetric nature can be observed from Eq. 3.1. As for the unboundedness, given any distribution P, with more than 1 event, if any entry of Q is equal to 0 when the corresponding entry in P is not, this will attain a divergence of ∞. We consider instead a symmetrised version, the *Jensen-Shannon Divergence* [61–63].

$$\begin{aligned} JS_{\mathcal{C}}(P,Q) &= \frac{1}{2}\left[KL_{\mathcal{C}}\left(P\middle|\frac{P+Q}{2}\right) + KL_{\mathcal{C}}\left(Q\middle|\frac{P+Q}{2}\right)\right] \\ &= H_{\mathcal{C}}\left(\frac{P+Q}{2}\right) - \frac{(H_{\mathcal{C}}(P)+H_{\mathcal{C}}(Q))}{2} \end{aligned} \tag{3.2}$$

In fact, the square root of this quantity is a metric. It is symmetric, it is zero only when the distibutions are equal and it obeys the triangle inequality [61–63]. Not only that, it is also bounded between 0 and 1.

These were later extended to be used in the field of Quantum Mechanics (QM) to measure the distance between quantum states or density operators, which generalise probability distributions [61]. A density operator ρ is by definition a positive semidefinite matrix whose (real, non-negative) eigenvalues add up to 1 [13, 40]. The quantum form of the quantities above are [33, 61–63]:

$$H_{\mathcal{Q}}(\rho) = -Tr[\rho\log_2(\rho)] = -\sum_{\gamma_i\in\sigma(\rho)}\gamma_i\log_2(\gamma_i) \tag{3.3}$$

$$KL_{\mathcal{Q}}(\rho|\phi) = Tr[\rho(\log_2(\rho)-\log_2(\phi))] \tag{3.4}$$

$$\begin{aligned} JS_{\mathcal{Q}}(\rho,\phi) &= \frac{1}{2}\left[KL_{\mathcal{Q}}\left(\rho\middle|\frac{\rho+\phi}{2}\right) + KL_{\mathcal{Q}}\left(\phi\middle|\frac{\rho+\phi}{2}\right)\right] \\ &= H_{\mathcal{Q}}\left(\frac{\rho+\phi}{2}\right) - \frac{(H_{\mathcal{Q}}(\rho)+H_{\mathcal{Q}}(\phi))}{2} \end{aligned} \tag{3.5}$$

where $\sigma(\rho)$ is the set of eigenvalues of ρ, including repeated eigenvalues and arranged such that $\gamma_i \leq \gamma_{i+1}$. Equation 3.3 and 3.5 are known respectively as the von Neumann entropy and the quantum Jensen-Shannon divergence (QJSD), to make the distinction between the classical and quantum forms [61–63].

It may not be apparent how the left hand side equals the right hand side in Eq 3.3, so we shall explain this equality. The logarithm of a matrix is the infinite sum:

$$\log(\rho) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{(\rho - I)^k}{k}.$$

This is converted into base 2 by dividing by $\log(2)$. Given that we know ρ is a real symmetric, positive semi-definite matrix it is diagonalisable. This means that it can be decomposed as $\rho = VDV^{-1}$, where D is a diagonal matrix containing the eigenvalues, λ, as its entries and the columns of V are the eigenvectors. Using this in the logarithm we get $\log(\rho) = V \log(D) V^{-1}$. Now consider our expression.

$$\begin{aligned}
-Tr[\rho \log_2(\rho)] &= -Tr[(VDV^{-1})(V \log_2(D) V^{-1})] \quad \text{using } \rho = VDV^{-1} \\
&= -Tr[VD \log_2(D) V^{-1}] \quad \text{using } V^{-1}V = I \\
&= -Tr[(D \log_2(D) V^{-1})V] \\
&= -Tr[D \log_2(D)] = -\sum_{\gamma_i \in \sigma(\rho)} \gamma_i \log_2(\gamma_i)
\end{aligned}$$

These quantities, $H_{\mathcal{Q}}(\rho), KL_{\mathcal{Q}}(\rho|\phi)$ and $JS_{\mathcal{Q}}(\rho, \phi)$, inherit the same properties as their classical counterparts and, in particular, the square root of the QJSD is a metric. It is still symmetric, which can be seen from the definition. The terms $KL_{\mathcal{Q}}(\rho|\sigma)$ are still non-negative and equal to zero if and only if the two density matrices are equal, $\rho = \sigma$ [61, 63–66]. This last fact can be seen by applying Klien's inequality and using the function $t \log_2(t)$, as in [65, 66], which gives:

$$Tr[A \log_2(A) - A \log_2(B)] \geq Tr[A - B] = Tr[A] - Tr[B]$$

Where equality is obtained if and only if $A = B$. Which would make it equal zero, since our matrices are rescaled to have trace equal to one. Furthermore, this means that $\sqrt{JS_{\mathcal{Q}}}$ is zero only when the density matrices are equal. As $KL_{\mathcal{Q}}(\rho|\frac{\rho+\phi}{2})$ and $KL_{\mathcal{Q}}(\phi|\frac{\rho+\phi}{2})$ are non-negative this means that for $\sqrt{JS_{\mathcal{Q}}}$ to equal zero both parts must be equal to zero. Therefore, from above, we must have $\rho = \frac{\rho+\phi}{2} = \phi$. Finally, it satisfies the triangle inequality [62].

Although these tools originated in QM and quantum information theory [61–63] they have started to be applied to graphs and networks. As already discussed in Section 1.3, the authors in [13] use the QJSD to determine which layers of a multilayer network to fuse together in order to optimise the number of layers needed to represent their system. In [44] they develop a method to compare the structural differences of networks, part of which uses the classical JSD. Rossi et al. [67] use the QJSD to characterise graph symmetries through quantum walks and in other papers [39, 40] has been used to study complex networks. Later in this chapter we show how the entropy and QJSD of graphs is related to the clustering structure of the networks. The rest of the work in this thesis

presents a new way of fusing multiple networks and producing one network that is as close as possible to the clustering structure of all of the given input network by using the QJSD. For now we cover the definitions and the re-formulation to apply these quantum quantities to graphs as in [13].

Definition 3.1. A *weighted graph* on n vertices can be represented by a real, symmetric $n \times n$ matrix with non-negative entries $G = (G_{ij}) \in Sym_{n\times n}(\mathbb{R}^{\geq 0})$ with zero-diagonal. Namely, there is an edge with weight $G_{ij} \neq 0$, and no such edge if $G_{ij} = 0$. This definition can represent unweighted graphs (G_{ij}=1 for all edges, by convention) and undirected graphs ($G_{ij} = G_{ji}$ for all i, j), as well as weighted self-loops ($G_{ii} \neq 0$) but not multi-edges. We restrict ourselves to positive edge weights, so that the associated Laplacian matrix (see below) is positive semidefinite.

From this point on we implicitly identify a graph and its adjacency matrix G. When studying graphs, the associated Laplacian matrix (defined below) is an important object to consider. It has been shown that its spectral properties (eigenvalues and eigenvectors) are closely related to structural properties of the graph, in particular its clustering structure. For a simple example, note that the multiplicity of the 0 eigenvalue equals the number of connected components of the network [68,69]. The Laplacian eigenvalues can be used to estimate the number of clusters in a graph and the eigenvectors to find clusters (eigengap and spectral clustering [53,54]); they can also be used to optimally embed the network into n dimensions (spectral embedding [54,70]) among other applications.

Definition 3.2. The *degree* matrix, D_G of an undirected, positively weighted graph with adjacency matrix $G = (G_{ij})$ is defined as:

$$D_{G,ij} = \begin{cases} \sum_i g_{ik} & \text{for } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

The *Laplacian matrix* for a graph G is then defined as:

$$L(G) = D_G - G$$

Recall that an $n \times n$ symmetric matrix A is positive semidefinite if

$$x^T Ax \geq 0 \qquad \forall x \in \mathbb{R}^n$$

We next show that the graph Laplacian is a positive semidefinite matrix. The proof is included so we can easily see how the *rescaled Laplacian*, which will be defined shortly, is also a positive semidefinite matrix.

Lemma 3.3. *The Laplacian matrix, as defined in Definition 3.2, is a positive semidefinite matrix.*

Proof. Given a graph G, set $G_{(u,v)}$ to be the subgraph with only the edge $(u,v) \in E(G)$, on the same number of vertices. This graph G, and its adjacency matrix, can then be written as the summation of all the adjacency matrices corresponding to the subgraphs with one edge, $G = \sum_{(u,v)\in E(G)} G_{(u,v)}$. The laplacian, is linear with respect to addition, shown in Appendix A and so can also be written as a summation of laplacians $L(G) = \sum_{(u,v)\in E(G)} L(G_{(u,v)})$. Then:

$$x^T L(G)x = \sum_{(u,v)\in E(G)} w_{uv}(x_u - x_v)^2$$

where w_{uv} is the weight of the edge, which, as they are are all non-negative, means this quantity is non-negative. Therefore it is a positive semidefinite matrix. □

Next we show that we can normalise the Laplacian so that the sum of its eigenvalues is 1, that is, it becomes a density operator. Recall that the trace of a matrix is equal to the sum of its eigenvalues [71]. Then:

Definition 3.4. The *rescaled Laplacian matrix* is defined as

$$\bar{L}(G) = \frac{L(G)}{Tr(L(G))} = \frac{D_G - G}{\sum_{ij} G_{ij}}$$

To see that this is still a positive semidefinite matrix, note that we have only divided by a non-negative quantity. Therefore the proof remains the same as the one above for the laplacian, but just with smaller positive edge weights. All in all, the rescaled Laplacian becomes a density operator. With these results the entropy of a graph and the QJSD between them can be calculated using the associated rescaled laplacian matrix. To be explicit, we re-state the definitions of entropy, and of Jensen-Shannon distance, for graphs.

Definition 3.5. Given a graph represented by an adjacency matrices G, we define its entropy as:

$$\begin{aligned} H_{\mathcal{N}}(G) &= -\sum_{\lambda_i\in\sigma(\bar{L}(G))}^{n} \lambda_i \log_2(\lambda_i) \\ &= -\sum_{\lambda_i\in\sigma(L(G))}^{n} \frac{\lambda_i}{\sum_j \lambda_j} \log_2\left(\frac{\lambda_i}{\sum_j \lambda_j}\right) \end{aligned} \tag{3.6}$$

Using $\sigma(\bar{L}(G))$ to denote the set of eigenvalues of $\bar{L}(G)$ which are ordered such that $\lambda_i \leq \lambda_{i+1}$. To define the Jensen-Shannon distance between two graphs we use the following.

Proposition 3.6. *Given two graphs represented by adjacency matrices G_1 and G_2 then the following is equivalent*

$$H_{\mathcal{Q}}\left(\frac{\bar{L}(G_1)+\bar{L}(G_2)}{2}\right) = H_{\mathcal{N}}(g_2G_1+g_1G_2)$$

where $g_1 = \sum_{i,j} G_{1,ij}$ and $g_2 = \sum_{i,j} G_{2,ij}$.

Proof. Consider the following:

$$\begin{aligned}
\frac{\bar{L}(G_1)+\bar{L}(G_2)}{2} &= \frac{1}{2}\left[\frac{D_{G_1}-G_1}{g_1}+\frac{D_{G_2}-G_2}{g_2}\right] \\
&= \frac{1}{2}\left[\frac{g_2D_{G_1}-g_2G_1+g_1D_{G_2}-g_1G_2}{g_1g_2}\right] \\
&= \frac{D_{g_2G_1+g_1G_2}-(g_2G_1+g_1G_2)}{2g_1g_2} \\
&= \bar{L}(g_2G_1+g_1G_2)
\end{aligned}$$

The last line follows from noting that the numerator is the laplacian for the matrix $g_2G_1+g_1G_2$. Where the sum all of the entries is equal to $2g_1g_2$ and therefore dividing by this quantity gives the rescaled laplacian. This matrix $g_2G_1+g_1G_2$ can then have its entropy calculated as per Definition 3.5. □

Definition 3.7. Given two graphs represented by adjacency matrices G_1 and G_2, we define their their Jensen-Shannon distance as:

$$JS_{\mathcal{N}}(G_1,G_2) = H_{\mathcal{N}}(g_2G_1+g_1G_2) - \frac{1}{2}(H_{\mathcal{N}}(G_1)+H_{\mathcal{N}}(G_2)) \tag{3.7}$$

it can also be noted from our previous discussion, that the quantum $JS_{\mathcal{Q}}$, and therefore the network $JS_{\mathcal{N}}$, are equal to zero if and only if the two density matrices are equal. For networks, two rescaled laplacian matrices are equal if and only if the original laplacian are scalar multiples of one another. To illustrate this further, consider Figure 3.1. These are two non-isomorphic graphs taken from [72] with co-spectral laplacians, that is, they have the same set of laplacian eigenvalues. Because of this, they achieve the same value for their entropy, (namely 3.707). If we were using the classical version, their distance $\sqrt{JS_C}$ would be zero, despite the fact they are distinct (non-isomorphic) graphs. In fact, the same issue would occur if we were to use the standard Euclidean distance between the eigenvalues. Any non-isometric co-spectral graphs, graphs that can not be transformed from one to the other by a function but both have the same set of laplacian eigenvalues, would be distance zero apart which would mean we would no longer have a metric. However, with the network definition, they have a non-zero distance $\sqrt{JS_N} = 0.1295$. This is because it also considers the average of the two density matrices and the change in eigenvalues is not linear with respect to matrix addition. Similarly, even though they have the same spectrum, their distance to a third graph are not necessarily equal. For

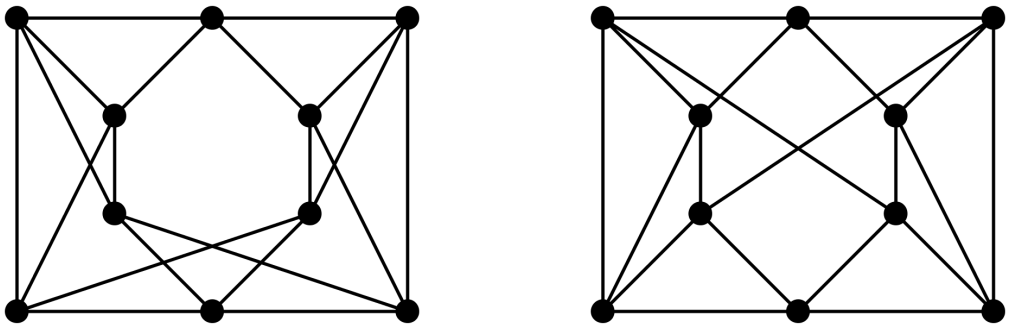

FIGURE 3.1: Two non-isomorphic, co-spectral graphs (same Laplacian eigenvalues) have the same entropy. Hence the Jenssen-Shannon divergence, $\sqrt{JS_{\mathcal{C}}}$, between these two graphs is zero. However, their Jenssen-Shannon distance, $\sqrt{JS_{\mathcal{N}}}$, is 0.1295, picking up on these finer details and subtly in structure.

example their distances to the ring graph, on the same number of verticies, are 0.3183 and 0.3295 respectively.

3.2 Example Networks

Having described how to adapt the classical definition of entropy to networks, in this section we present some results on the entropy of some common families of graphs/networks and operations.

Definition 3.8. The following are common (undirected, unweighted) graphs that appear in graph theory. (note that we labels vertices $1, 2, \ldots n$ and identify each vertex with its label i):

- The graph with n vertices where every pair of distinct vertices is connected by an edge is called the *complete* graph on n vertices, written K_n [73, 74].
- The graph with $n + m$ vertices and an edge between i and j for every $1 \leq i \leq n$ and $n + 1 \leq j \leq m + n$ is the *complete bipartite* graph $B_{n,m}$ [73, 74].
- The *star* graph with n vertices is the graph $S_n = B_{1,n-1}$ [73, 74].
- The *path* graph, P_n, with n vertices is the graph with edges $\{(i, i + 1)\}, 1 \leq i < n$ [73, 74].
- The *ring* graph with n vertices, R_n is the graph, P_n, with the extra edge $\{(1, n)\}$ [73, 74].

Lemma 3.9. *The following are the eigenvalues of the laplacian for the graphs stated in the previous definition.*

- *The laplacian eigenvalues for* K_n *are* 0 *with multiplicity* 1 *and* n *with multiplicity* $n-1$.
- *The laplacian eigenvalues for* $B_{n,m}$ *are* 0 *with multiplicity* 1, n *with multiplicity* $m-1$, m *with multiplicity* $n-1$ *and* $n+m$ *with multiplicity* 1.
- *The laplacian eigenvalues for* S_n *are* 0 *with multiplicity* 1, 1 *with multiplicity* $n-2$ *and* n *with multiplicity* 1.
- *The laplacian eigenvalues for* R_n *are* $2-2\cos\left(\frac{2\pi}{n}k\right)$ *for* $0 \leq k \leq \frac{n}{2}$ *each with multiplicity* 2 *except for* $k=0$ *and, when* n *is even,* $k=\frac{n}{2}$.
- *The laplacian eigenvalues for* P_n *are the same as* R_{2n}.

With the eigenvalues for each of these graphs in hand, we now proceed to explicitly calculate their entropy.

Lemma 3.10. *The entropy of the complete graph* K_n *is* $H_{\mathcal{N}}(K_n) = \log_2(n-1)$.

Proof. The trace (sum of the eigenvalues) is equal to $n(n-1)$ hence the entropy:

$$\begin{aligned}
H_{\mathcal{N}}(K_n) &= -\sum_{\lambda_i \in \sigma(L_{K_n})} \frac{\lambda_i}{\sum \lambda_j} \log_2\left(\frac{\lambda_i}{\sum \lambda_j}\right) \\
&= -\sum_{i=2}^{n} \frac{n}{n(n-1)} \log_2\left(\frac{n}{n(n-1)}\right) \\
&= -\sum_{i=2}^{n} \frac{1}{(n-1)} \log_2\left(\frac{1}{(n-1)}\right) \\
&= -\frac{(n-1)}{(n-1)} \log_2\left(\frac{1}{(n-1)}\right) \\
&= \log_2(n-1).
\end{aligned}$$

□

Lemma 3.11. *The entropy of the bipartite graph $B_{n,m}$ is* $H_{\mathcal{N}}(B_{n,m}) = 1+\frac{1}{2}\log_2(mn)+\frac{1}{2m}\log_2(n)+\frac{1}{2n}\log_2(m)-\frac{n+m}{2nm}\log_2(n+m)$.

Proof. The trace (sum of the eigenvalues) is equal to $2nm$ hence the entropy:

$$\begin{aligned}
H_{\mathcal{N}}(B_{n,m}) &= -\sum_{\lambda_i\in\sigma(L_{B_{n,m}})}^{n+m}\frac{\lambda_i}{\sum\lambda_j}\log_2\left(\frac{\lambda_i}{\sum\lambda_j}\right)\\
&= -\frac{n(m-1)}{2nm}\log_2\left(\frac{n}{2mn}\right)-\frac{m(n-1)}{2nm}\log_2\left(\frac{m}{2mn}\right)-\frac{n+m}{2nm}\log_2\left(\frac{n+m}{2mn}\right)\\
&= -\frac{1}{2}\log_2\left(\frac{n}{2mn}\right)+\frac{n}{2nm}\log_2\left(\frac{n}{2mn}\right)-\frac{1}{2}\log_2\left(\frac{m}{2mn}\right)+\frac{m}{2nm}\log_2\left(\frac{m}{2mn}\right)\\
&\qquad -\frac{n+m}{2nm}\log_2\left(\frac{n+m}{2mn}\right)\\
&= \frac{1}{2}\log_2(4mn)+\frac{n}{2nm}\log_2\left(\frac{n}{2mn}\right)+\frac{m}{2nm}\log_2\left(\frac{m}{2mn}\right)-\frac{n+m}{2nm}\log_2\left(\frac{n+m}{2mn}\right)\\
&= 1+\frac{1}{2}\log_2(mn)+\frac{1}{2m}\log_2(n)+\frac{1}{2n}\log_2(m)-\frac{n+m}{2nm}\log_2(n+m)\,.
\end{aligned}$$

□

Corollary 3.12. *The entropy of the star graph S_n is* $H_{\mathcal{N}}(S_n) = 1+\log_2(n-1)-\frac{n}{2n-2}\log_2(n)$.

An exact analytic result for the entropy of the ring and path graph is much more involved so instead, we will compute it numerically. Table 3.1 below, summarises the formulas for the entropy of the graphs we could obtain. Figure 3.2 illustrates the growth of the entropies of the graphs K_n, S_n, P_n and R_n as n, the number of vertices, grows. Here we have an indication of how entropy captures the structure and disorder of the graph. The line for the complete graph is always above any of the other graphs. This is to be expected as every node is connected to every other node making the graph have no structure whatsoever and contains no information on how it should be clustered. Meanwhile the ring graph and the path graph have more of a clear structure, reducing their entropy. The path graph is always slightly below because it has one less edge, making a simpler structure. We see the lowest entropy of the graphs that we have considered are the star graphs. This is because, apart from one node, all the nodes are connected to the same node. This has the simplest structure to describe with no ambiguity of how they should be clustered together, resulting in having the lowest entropy. Next we show results about the composition of disjoint graphs.

Graph	Trace	Entropy
complete	$n(n-1)$	$\log_2(n-1)$
bipartite	$2nm$	$1+\frac{1}{2}\log_2(mn)+\frac{1}{2m}\log_2(n)+\frac{1}{2n}\log_2(m)-\frac{n+m}{2nm}\log_2(n+m)$
star	$2n-2$	$1+\log_2(n-1)-\frac{n}{2n-2}\log_2(n)$

TABLE 3.1: A table of the formulas calculated for the entropy of the respective graphs.

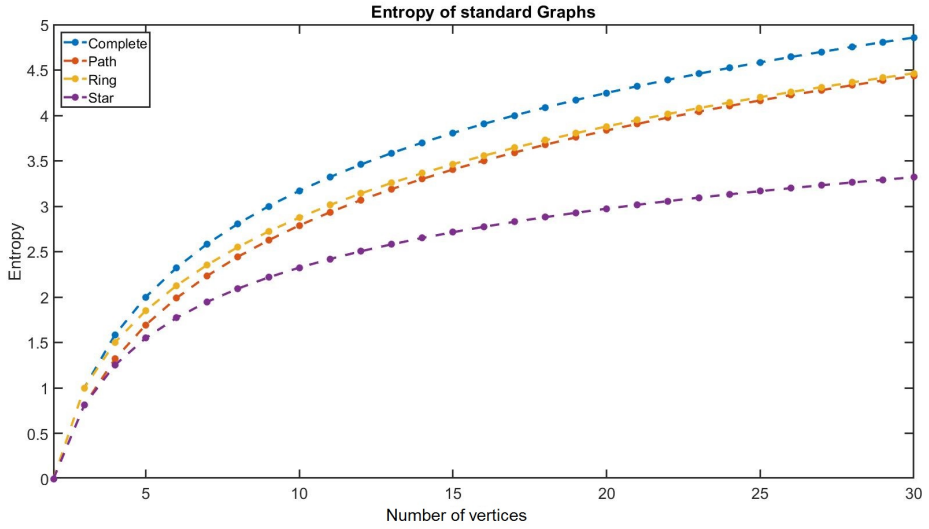

FIGURE 3.2: This figure shows the entropies of some of the standard graphs discussed as the number of vertices increase. Specifically the complete graph K_n, path graph P_n, the ring graph R_n and the star graph S_n

For the upcoming calculation, we recall that if a graph G is the disjoint union of two graphs, G_1, G_2, then the spectrum of G is the union of the spectrum of G_1 and G_2. This follows from the fact that the characteristic polynomial of a block diagonal matrix is the product of the respective characteristic polynomials. We also require to calculate the entropy of a normalised set of non-negative numbers. Therefore we make the following definition.

Definition 3.13. Given a set of nonnegative numbers, $\{a_1, \ldots, a_n\}$, we define the *set entropy* to be $\tilde{h}(\{a_1, \ldots, a_n\}) = -\sum_{i=1}^{n} \frac{a_i}{\sum_j a_j} \log_2(\frac{a_i}{\sum_j a_j})$ with the convention of $0 \log_2(0) = 0$.

Theorem 3.14. *Let G be the disjoint union of two non-empty graphs G_1, G_2 then the entropy is given by $H_{\mathcal{N}}(G) = \frac{1}{\sum_{i=1}^{2} T_i} \left[\sum_{i=1}^{2} T_i H_{\mathcal{N}}(G_i)\right] + \tilde{h}(\{T_1, T_2\})$ where T_1, T_2 are the sum of the eigenvalues for G_1 and G_2 respectively.*

Proof. Let $H_{\mathcal{N}}(G_1) = -\sum_{i=1}^{n} \frac{\lambda_i}{T_1} \log_2(\frac{\lambda_i}{T_1})$ and $H_{\mathcal{N}}(G_2) = -\sum_{j=1}^{n} \frac{\rho_j}{T_2} \log_2(\frac{\rho_j}{T_2})$. Thanks to the previous lemma the sum of the eigenvalues in G is $T_1 + T_2$. Therefore the entropy

is given by:

$$\begin{aligned}
H_{\mathcal{N}}(G) &= -\left[\sum_{i=1}^{n} \frac{\lambda_i}{T_1+T_2} \log_2\left(\frac{\lambda_i}{T_1+T_2}\right)\right] - \left[\sum_{j=1}^{n} \frac{\rho_j}{T_1+T_2} \log_2\left(\frac{\rho_j}{T_1+T_2}\right)\right] \\
&= -\frac{T_1}{T_1+T_2}\left[\sum_{i=1}^{n} \frac{\lambda_i}{T_1} \log_2\left(\frac{\lambda_i}{T_1} \cdot \frac{T_1}{T_1+T_2}\right)\right] - \frac{T_2}{T_1+T_2}\left[\sum_{j=1}^{n} \frac{\rho_j}{T_2} \log_2\left(\frac{\rho_j}{T_2} \cdot \frac{T_2}{T_1+T_2}\right)\right] \\
&= -\frac{T_1}{T_1+T_2}\left[\sum_{i=1}^{n} \frac{\lambda_i}{T_1} \log_2\left(\frac{\lambda_i}{T_1}\right) + \sum_{i=1}^{n} \frac{\lambda_i}{T_1} \log_2\left(\frac{T_1}{T_1+T_2}\right)\right] \\
&\qquad - \frac{T_2}{T_1+T_2}\left[\sum_{j=1}^{n} \frac{\rho_j}{T_2} \log_2\left(\frac{\rho_j}{T_2}\right) + \sum_{j=1}^{n} \frac{\rho_j}{T_2} \log_2\left(\frac{T_2}{T_1+T_2}\right)\right] \\
&= \frac{T_1}{T_1+T_2}\left[H_{\mathcal{N}}(G_1) - \log_2\left(\frac{T_1}{T_1+T_2}\right)\right] + \frac{T_2}{T_1+T_2}\left[H_{\mathcal{N}}(G_2) - \log_2\left(\frac{T_2}{T_1+T_2}\right)\right] \\
&= \frac{1}{\sum_{i=1}^{2} T_i}\left[\sum_{i=1}^{2} T_i H_{\mathcal{N}}(G_i)\right] + \tilde{h}(\{T_1, T_2\}).
\end{aligned}$$

□

Note that $T_i = \sum_{j,k} g_{jk}$ is also the sum of all the edge weights. The above theorem can then also be extended, by induction, to include multiple disjoint components, not just two.

Corollary 3.15. *Let G be the disjoint union of c non-empty graphs, $G_1, \ldots, G_c$ then the entropy is given by $H_{\mathcal{N}}(G) = \frac{1}{\sum_{i=1}^{c} T_i}\left[\sum_{i=1}^{c} T_i H_{\mathcal{N}}(G_i)\right] + \tilde{h}(\{T_1, \ldots, T_c\})$.*

Corollary 3.16. *If $G = G_j$, for all j, and not the empty graph, then the entropy of the disjoint union of $G_1, \ldots, G_c$ is given by $H_{\mathcal{N}}(G) + log_2(c)$*

Proof.

$$\begin{aligned}
H_{\mathcal{N}}(G) &= \frac{1}{\sum_{i=1}^{c} T_i}\left[\sum_{i=1}^{c} T_i H_{\mathcal{N}}(G_i)\right] + \tilde{h}(\{T_1, \ldots, T_c\}) \\
&= \frac{1}{cT_i}\left[cT_i H_{\mathcal{N}}(G_i)\right] - \sum_{i=1}^{c} \frac{T_i}{cT_i} \log_2\left(\frac{T_i}{cT_i}\right) \\
&= H_{\mathcal{N}}(G_i) - \frac{c}{c} \log_2\left(\frac{1}{c}\right) \\
&= H_{\mathcal{N}}(G_i) + \log_2(c).
\end{aligned}$$

□

3.3 Network Entropy Examples

In this section we discuss examples and results that help us gain insight of what entropy captures in a network. Consider a graph made up of c disjoint complete graphs of m vertices each. (This is the ideal case of a network with $n = cm$ vertices and c identical clusters.) The laplacian of G will be a block diagonal matrix whose eigenvalues is the union of the eigenvalues of each block [68, 69]. Each block is a complete graph K_m with laplacian eigenvalues 0 with multiplicity 1 and m with multiplicity $m-1$ [68, 69]. Therefore the spectrum (set of eigenvalues) of the laplacian of G is 0 with multiplicity c and m with multiplicity $c(m-1)$. Normalising these such they sum to one they become 0 and $\frac{1}{c(m-1)}$. Therefore calculating the entropy (from Eq. 1.2):

$$\begin{aligned} H_{\mathcal{N}}(G) &= -\sum_{i=1}^{n} \lambda_i \log_2(\lambda_i) \\ &= -c[0 \log_2(0)] - c(m-1)\left[\frac{1}{c(m-1)} \log_2\left(\frac{1}{c(m-1)}\right)\right] \\ &= 0 - \log_2\left(\frac{1}{c(m-1)}\right) \\ &= \log_2(cm - c) \\ &= \log_2(n - c). \end{aligned} \tag{3.8}$$

This shows that the number of clusters that naturally arise within the network, encoded by the eigenvalues, is linked to the entropy and decreases as the number of clusters increase. Clearly this increases with the number of nodes, n, as more nodes leads to more possibilities about how they should be clustered together and therefore more uncertainty. Looking at this closer, the number of clusters, c, is bounded between $1 \leq c \leq n-1$. These bounds correspond to when every node is grouped into one cluster and the other being when only two are grouped together with every other node on its own. For the above example, this means:

$$0 = \log_2(1) \leq H_{\mathcal{N}}(G) \leq \log_2(n-1) \tag{3.9}$$

The bounds seen above, maximal value of $\log_2(n-1)$ and minimal value of 0, agrees with results in the literature [32, 33, 35]. Equation 3.8 also reveals that two graphs, in our simple example, shall have the same entropy if the difference between the total number of nodes and clusters are equal. For example the graph K_5 will have the same entropy as the graph that is the disjoint union of two copies of K_3.

Note that entropy is not only dependent on the number, or density, of edges, but their location within the graph. For a simple example, consider the path graph P_n, and the star graph S_n, on the same number n of vertices. Both of these graphs have the

same number of edges, $n-1$, however by looking back at Figure 3.2 these graphs have asymptotically different entropies, showing that entropy is not only effected by the number of edges *but* how they are allocated. This is further supported by the example graphs in Figure 3.3. Here 4 graphs are shown all with the same number of vertices and all having 10 edges, apart from (C) which has 8. As can be seen the different possible allocation of edges produces a range of values for the entropy. The effect of separate components is apparent upon comparison of (A) and (D) but more important is the comparison between (C) and (B). Here the ring graph, (C), has 2 edges less than that of (B) but the entropy of (B) is significantly lower, hence showing that entropy is not only effected by edge density but also its allocation.

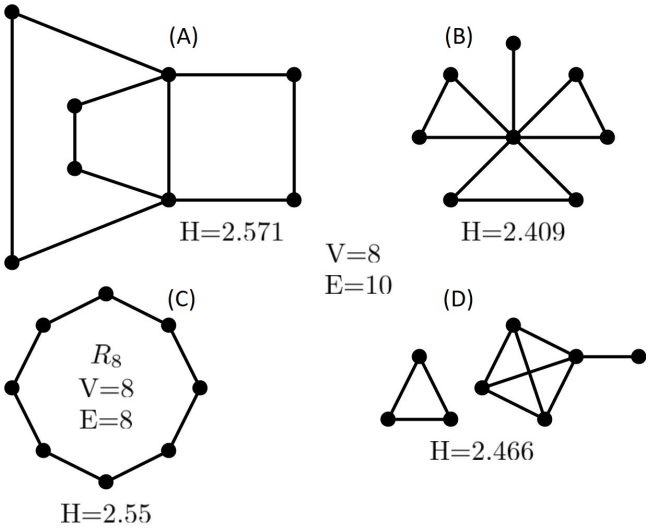

FIGURE 3.3: Examples of how entropy is not only related to edge density but also location. All of which have the same number of vertices, 8 and A, B, D having 10 edges and C only 8 which is the Ring graph. We use H to denote the entropy for each graph.

A mathematical measure of how well connected a subgraph (cluster) of a network is to the rest of the network is to measure the RatioCut [54]. The closer the value is to 0 the easier that subset is to remove from the network, that is, the fewer number of edges, relatively to its size, are needed to disconnect a subgraph from the rest of the network. Below we show evidence that there is a link between the entropy of the network and the RatioCut. Formally let G be a graph (network) with vertex set V and edge set E. Let S be a subset of V representing the vertices of a subgraph or cluster and let $\bar{S}$ be the compliment of V (those not in V). The *boundary*, ∂S, is the sum of edge weights, w_{ij}, where one end of the edge is in the subset and the other is not, that is,

$$\partial S = \sum_{\{i \in S, i \in \bar{S}\}} w_{ij}.$$

Then, given disjoint clusters of the node in a graph, S_i, the RatioCut [54] is defined as:

$$RatioCut(S_1, \ldots, S_k) = \frac{1}{2} \sum_{i=1}^{k} \frac{\partial S_i}{|S_i|}.$$

Where $|S_i|$ is the number of vertices in the subset. The $\frac{1}{2}$ is included as each edge would be counted twice. In the unweighted case, the upper bound is given by:

$$\begin{aligned} RatioCut(S_1, \ldots, S_k) &\leq \frac{1}{2} \sum_{i=1}^{k} \frac{|S_i|(|V| - |S_i|)}{|S_i|} = \frac{\sum_{i=1}^{k}(|V| - |S_i|)}{2} \\ &= \frac{k|V| - \sum_{i=1}^{k} |S_i|}{2} = \frac{k|V| - |V|}{2} = \frac{(k-1)|V|}{2} \end{aligned}$$

In the case of $k = 2$ clusters, when this quantity is considered over of all possible subsets of $V(G)$ the minimum is referred to as the Cheeger constant of the graph G. This constant is computationally hard to compute but can be estimated from the second eigenvalue of the graph laplacian via the Cheeger inequality [69, 73, 75, 76]. Further, by using a relaxation of the problem, the clustering that gives the RatioCut closest to this estimate can be found from using the eigenvector associated with the second smallest eigenvector. This method can also be used for $k > 2$ clusters and forms the basis of spectral clustering [54]. To explore the relation between entropy and RatioCut we conducted the following numerical experiment. An initial network was formed consisting of multiple blocks of K_n, the complete graph on n vertices where n has been chosen randomly, and then a random number of edges was added to the network. The RatioCut was then calculated, with respect to the original blocks as the clusters and the increase in entropy, as a percentage of maximum possible increase. This was then repeated 200 times and the results plotted against each other. After this the whole experiment was then run again multiple times with different selections of K_n, in each setting, the RatioCut was divided by its upper bound so that it was bounded between 0 and 1. A selection of the results are shown in Figure 3.4 all showing the same conclusion, that the entropy and the RatioCut increase proportionally as edges between clusters are added and as the distinct clusters disintegrate.

Let us now bring our consideration of entropy back to eigenvalues. Eigengap [54] is a way of inferring the number of clusters that a network naturally contains, this is done by looking for the largest gap between two consecutive eigenvalues, which are arranged in increasing order. When plotted, eigenvalues take an approximate "S" shape which we can model using the Sigmoid function $S(x)$:

$$S(x) = \frac{1}{1 + e^{-a(x-c)}} \tag{3.10}$$

We see $S(x)$ will go to 1 and 0 when x tends to $+\infty$ and $-\infty$ respectively and will equal $\frac{1}{2}$ when $x = c$. The c controls the location of this center and a controls the gradient

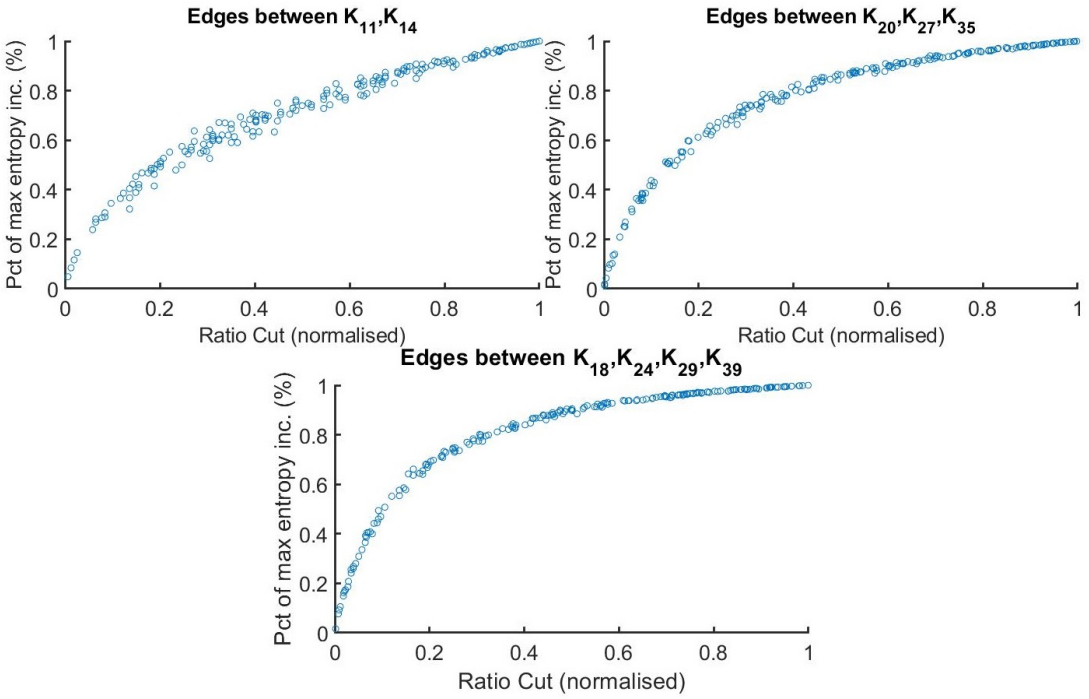

FIGURE 3.4: Examples of how entropy increases proportionally to the RatioCut of the clusters, for a selection of vertices and number of blocks. Each graph contains 200 data points where the original network has had a random number of edges added to it and the entropy and concordance calculated.

of the change over the center. (Note that this function is used in logistic regression problems for classifying objects [77, 78].) Here we use $S(X)$ it to model eigenvalues, the center being the number of clusters that there are in the network and the gradient eigengap/how well-defined those clusters are. Examples of these models are shown in Figure 3.5 for a variety of settings, note that each setting has been scaled such that the eigenvalues sum to 1, as they would when calculating entropy. To reiterate, this is an approximation of eigenvalues, in a real situation they would not necessarily for a smooth curve, but for our purpose here it is sufficient.

Now we have a model of all possible shapes of eigenvalues characterised by two parameters, a and c, and for each pair we can calculate the entropy. With a setting for a and c selected we use the value of $S(x)$ at $x = 1, \ldots, 20$, as the eigenvalues of our 'graph'. After normailising the eigenvalues to sum to 1 the entropy is calculate as we normally do in Eq. 3.6. We plotted the surface generated, Figure 3.6, by considering a over the range $[0, 2]$ and c over the range $[1, 20]$. Here we see two main effects. We see that as the number of clusters increase, c, the entropy decreases. This not only agrees with our intuition, more clusters means more structure resulting in a lower entropy, but it coincides with our analysis at the start of this subsection (Eq 3.8). The other thing we see is the effect of the gradient, a. As a increases, the eigengap for that cluster is becoming larger meaning that those clusters are becoming more well defined, the structure stronger and

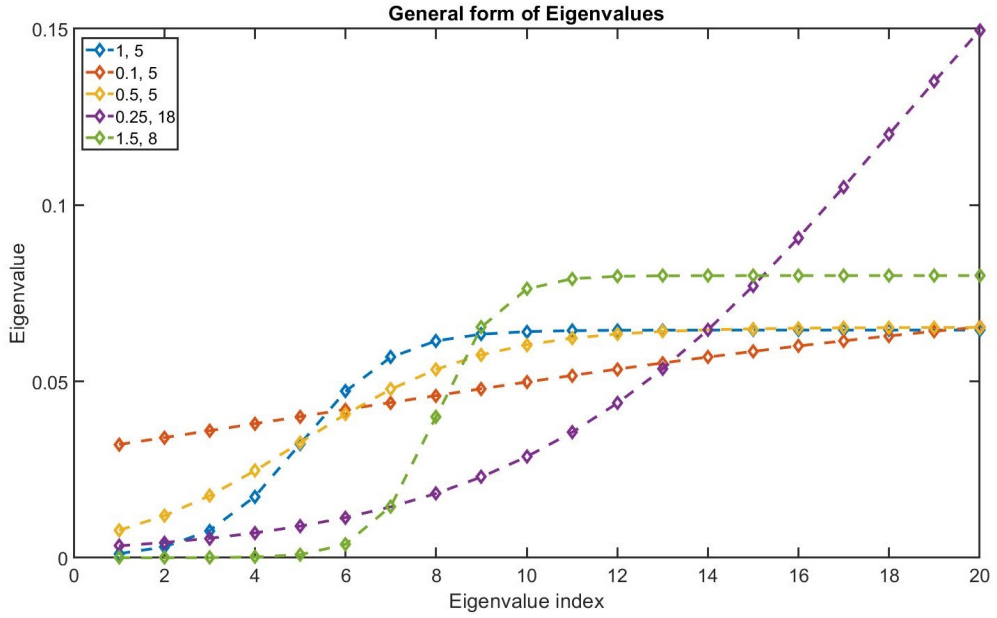

FIGURE 3.5: Modeling Eigenvalues using the Sigmoid function (Eq. 3.10). In the legend the first number is the setting for the gradient, how well defined the number of cluster are and the second value is the setting for the number of clusters. For each setting the eigenvalues were re-scaled such that they summed to 1, as they would be in the calculation for entropy.

more obvious, resulting in the entropy decreasing. This is further supported by Figure 3.7, which shows how the entropy changes for the same models but when one of the variables, a or c is fixed.

3.4 Network QJSD Examples

In this final section we illustrate how the QJSD quantifies the distance between networks with respect to structural differences. First we consider a rewiring of a highly clustered network and the effects on the entropy and the $\sqrt{JS_{\mathcal{N}}}$. We generated our initial network with distinct clusters using a Erdős-Rényi block model. In this model the nodes are preallocated into clusters and then the edges between two nodes, i and j, are generated according to the following scheme. If i and j are in *the same* cluster then the edge has a high probability of being generated, if they are *not* in the same cluster then the edge will have a very low probability of being generated.

As for the rewiring process, by rewiring we mean selecting a random existing edge, remove it, and then adding a new random edge, such that we are not creating self loops or multiple edges between two nodes. Given a sufficient number of rewires the structure of the network will completely change from its initial structure to, eventually, an entirely random structure (with fixed number of edges). We divided the rewiring process into two stages. The first 300 rewirings were done within the clusters, the new edge was

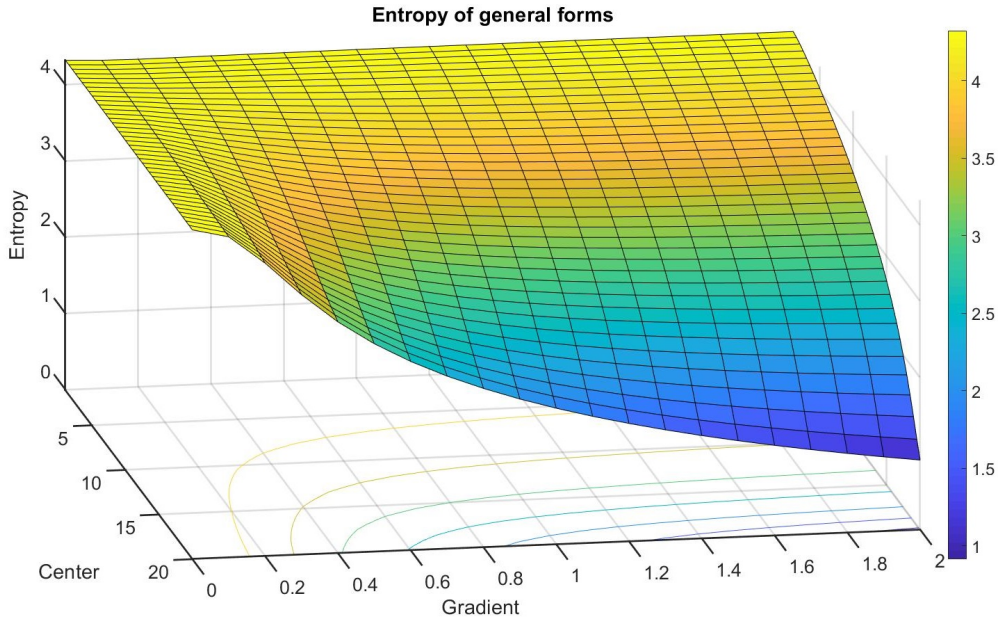

FIGURE 3.6: Entropy of sets of eigenvalues that have been generated using our Sigmoid model (Eq. 3.10), like those shown in Figure 3.5. The Centre is the location of the eigengap/the number of clusters that there are and the Gradient is how large the eigengap is/how well defined those clusters are.

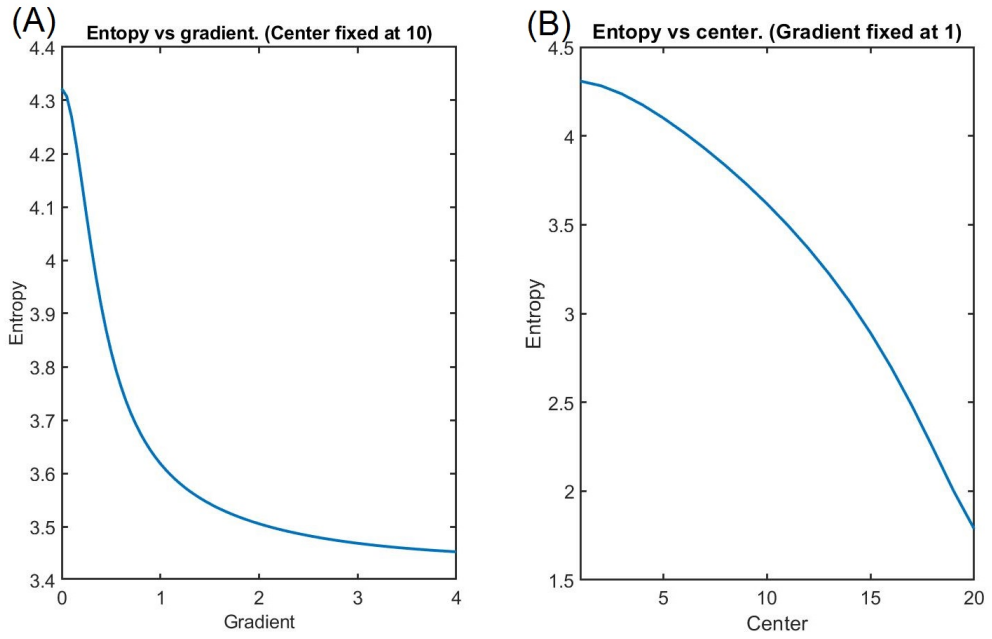

FIGURE 3.7: Entropy of sets of eigenvalues that have been generated using our Sigmoid model (Eq. 3.10), like those shown in Figure 3.5. (A) Shows the entropy when the centre is fixed ($c = 10$) and the gradient changes. (B) Shows the entropy when the gradient is fixed ($a = 1$) and the centre changes.

Rewiring Strategies	Internal	Global
Internal	0.1206	0.2009
Global	-	0.2023

TABLE 3.2: Given the distance matrix shown in Figure 3.8 (B) this can be broken into blocks corresponding to the different rewiring strategies. Then for each block we calculate the average non-trivial distance, the results of which are shown in this table.

added within the same cluster as the edge that was removed. The last 300 rewirings had no such restrictions so the new edge could be added anywhere. This was done to see the effect of the rewiring restrictions on the entropy of the network, particularly how it evolved, which are shown in Figure 3.8.

Figure 3.8 (A) shows the initial network on 100 vertices with 412 edges, generated by a Erdős-Rényi block model, yielding a highly clustered network. This network was then subjected to 600 rewirings, according to the strategy above, and the QJSD between all the various stages of the rewiring computed. The results are shown in Figure 3.8 (B). Here, as expected, given a network at any stage, it is closest to those that are a small number of rewiring away and gets further away as more rewiring occurs. This is because within a few rewirings the structure of the network does not change that significantly. In Figure 3.8 (C) we show the entropy at each step of the rewiring process, including a moving average (using 20 points) to smooth out the curve. The fact that the entropy changes, further suggests that entropy is affected by edge allocation, but what is interesting to note is the effects of the rewiring restriction. Looking at the first 300 rewirings if the entropy was simply capturing the conductance or the isoperimetric ratio of the graph then we would expect to see a straight line as the connections between the clusters are not changing. However we can see that there are small changes in the entropy (mean: 6.4366, standard deviation: 3.1×10^{-2}) showing that entropy is capturing more that these quantities. But we can see it is heavily affected by the clustering of the network as when the restriction is dropped, the entropy rapidly increases as the highly clustered structure disintegrates (mean: 6.4606, standard deviation: 1.31×10^{-1}). If we look at the average non-trivial distances in the blocks corresponding to the different rewiring strategies, this disintegration is also apparent. Table 3.2 shows that the average distance increases (by 67%) when we switch from internal to global rewiring. This shows how when the overall cluster structure is similar and small scale changes occur their distance is smaller and when large scale changes occur this greatly increases the distance.

Another example of how QJSD captures this difference in clustering structure is shown in Figure 3.9. Here disjoint copies of K_{10} form an initial graph, blue for 2 copies and orange for 4. Then a random number of edges are added, between two of the blocks. Once this is done then the distance between this edited network and the original, $\sqrt{JS_{\mathcal{N}}}$, is computed, and the results plotted. The graph shows that as the structure changes, then the distance between the original and edited network increase respectively. The blue line stabilises because the network tends to the complete network, there are only

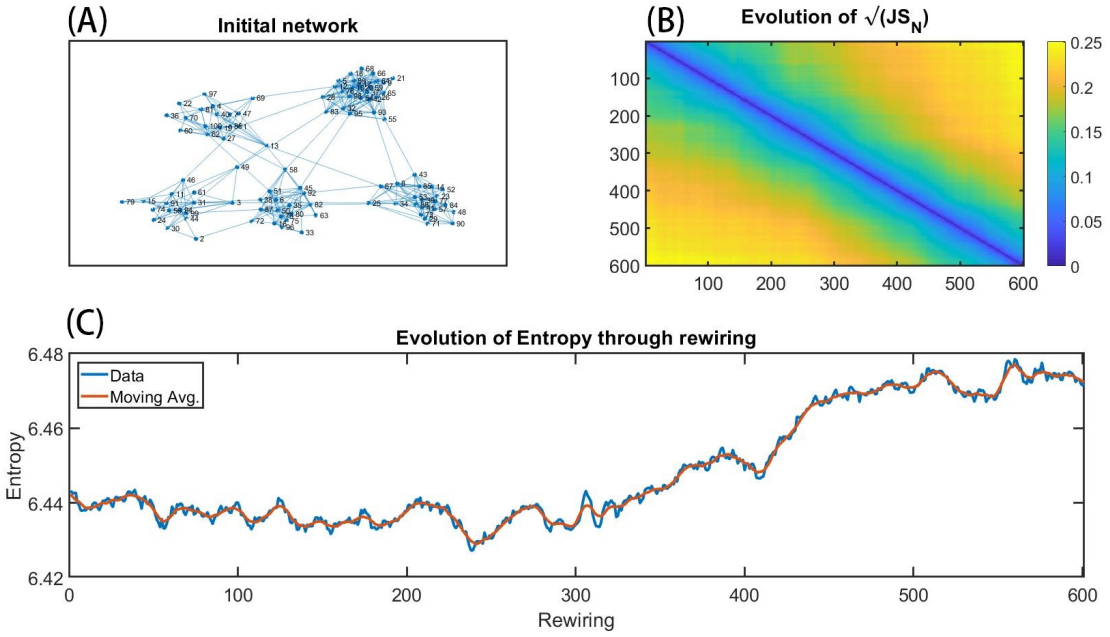

FIGURE 3.8: (A) Shows the initial network to be rewired, which was created using a Erdős-Rényi block model containing 100 nodes and 412 edges. (B) Shows a heatmap of the $\sqrt{JS_{\mathcal{N}}}$ distance between all the iterations during the rewiring process with the axes denoting the iteration. (C) At each iteration the entropy $H_{\mathcal{N}}$ was recorded and plotted. During the first 300 iterations, the rewirings were done such that the new edge was added to the same cluster it came from. After this point the restriction was dropped allowing the new edge to be added anywhere.

two blocks of K_{10} and the addition of 100 edges transforms it into K_{20}. As the orange line is on 4 blocks and the edges being added is restricted to being added between the same two blocks it does not stabilise. However if the restriction were removed and more than 100 edges were added it would eventually stabilise as the graph would become the complete graph K_{40}.

Let us also consider our sigmoid model from the previous section (Eq.3.10) and how we can analyse the QJSD with it. One feature that makes the analysis of the QJSD harder is that the eigenvalue of the addition is not the addition of the eigenvalues of the two matrices. However by imposing some conditions we can look at a subset of matrices where this is possible to help provide understanding. Assume we have two laplacians, G and $H \in M_n$ and now let us suppose they are simultaneously diagonalizable. That is there is an matrix S such that $G = S^{-1}ES$ and $H = S^{-1}FS$ where E and F are diagonal matrices, containing their eigenvalues. Note though that the eigenvalues may not be in the same order in both diagonal matrices, i.e in ascending order. However for the purpose of this discussion we are going to make the extra assumption that they are both arranged in ascending order. Altogether we have $\frac{1}{2}(G+H) = S^{-1}\frac{1}{2}(E+F)S$ meaning, in this case, that the eigenvalues of the average matrix is the average of the two sets of eigenvalues. Therefore, under these assumptions, we can easily calculate

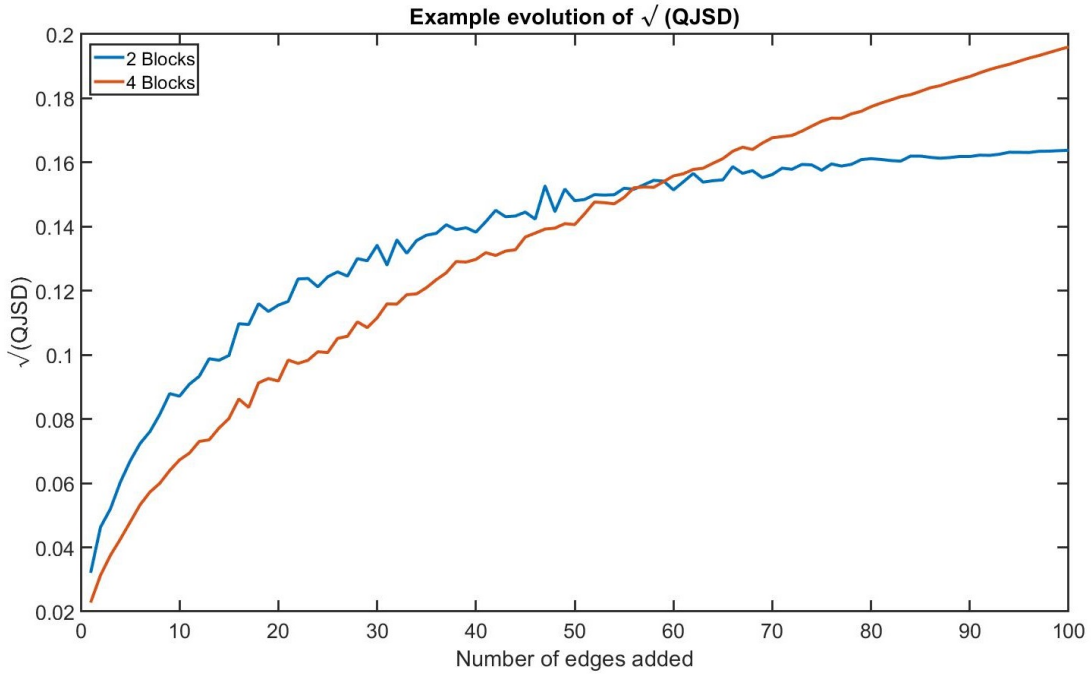

FIGURE 3.9: Here we start with graphs that are made of disjoint copies of K_{10} and then we add a number of edges randomly between two fixed clusters. The distance between the modified graphs and their originals were recorded and then plotted against the number of edges added. The blue line shows the change in $\sqrt{JS_{\mathcal{N}}}$ when we began with two blocks of K_{10} and the orange line is when begin with four blocks.

the QJSD between sets of eigenvalues that we have modeled with the sigmoid function (Eq.3.10). This is the square root of the entropy of the average set minus the average entropy of the two individual sets. Figure 3.10 shows the distance between a fixed set of eigenvalues characterized by our sigmoid model using $a = 0.9$ and $c = 8$, and the sets characterized by values on the x and y axis. This again shows QJSD is capturing and quantifying the structural information, encoded within the eigenvalues. We see that the distances get closer to zero as the structures become more and more similar, (as $a \to 0.9, c \to 8$), particularly the number of clusters have the greatest effect in reducing the distance. The only setting that achieves a value of zero is the one with the same settings and small values in the neighborhood around it. Similar results were achieved with other settings. This is also supported by Figure 3.11, where we calculate the distance between two forms as we vary one variable at a time, keeping the other fixed. The model we are computing the distance to is the same as before, the one charactised by ($a = 0.9$, $c = 8$).

Our final example to illustrate how QJSD captures the structural difference between two networks is using K-Nearest Neighbors (KNN) to "rewire" a network. Recall that KNN works by looking at the distance from the current vertex to all of the others and then only add a connecting edge to the closest K, which is selected by the user. Consider

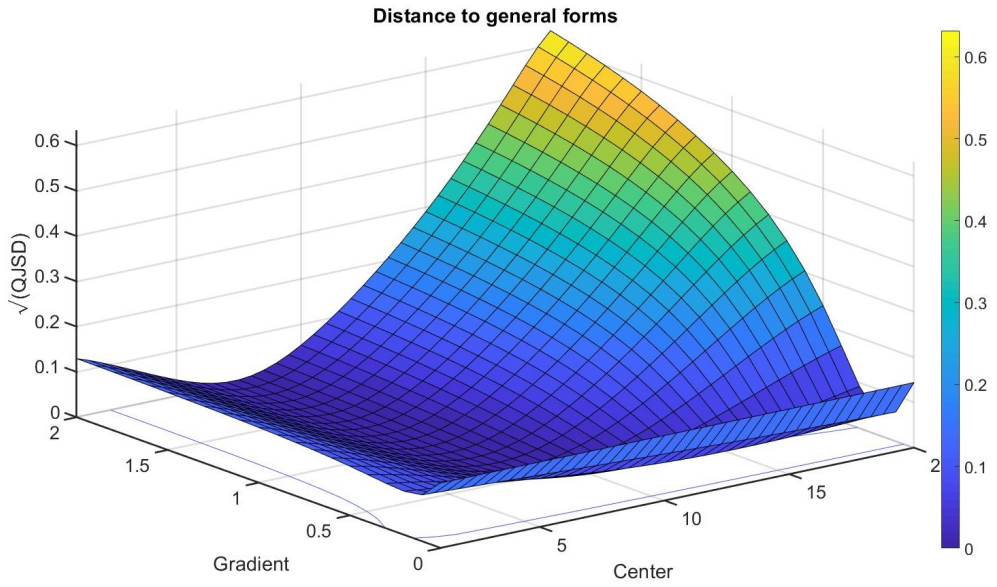

FIGURE 3.10: Using our sigmoid model (Eq.3.10) and some additional assumptions this figure show the $\sqrt{JS_{\mathcal{N}}}$ between all the general forms characterised by $a \in [0, 2], c \in [0, 20]$ and a fixed form $a = 0.9, c = 8$. As we can see the distance quickly tends to 0 as it approaches the same setting, this is the only location equal to zero. The neighbourhood around this is point has very small distances and we see that that achieving the same number of clusters is a main factor of reducing the distance.

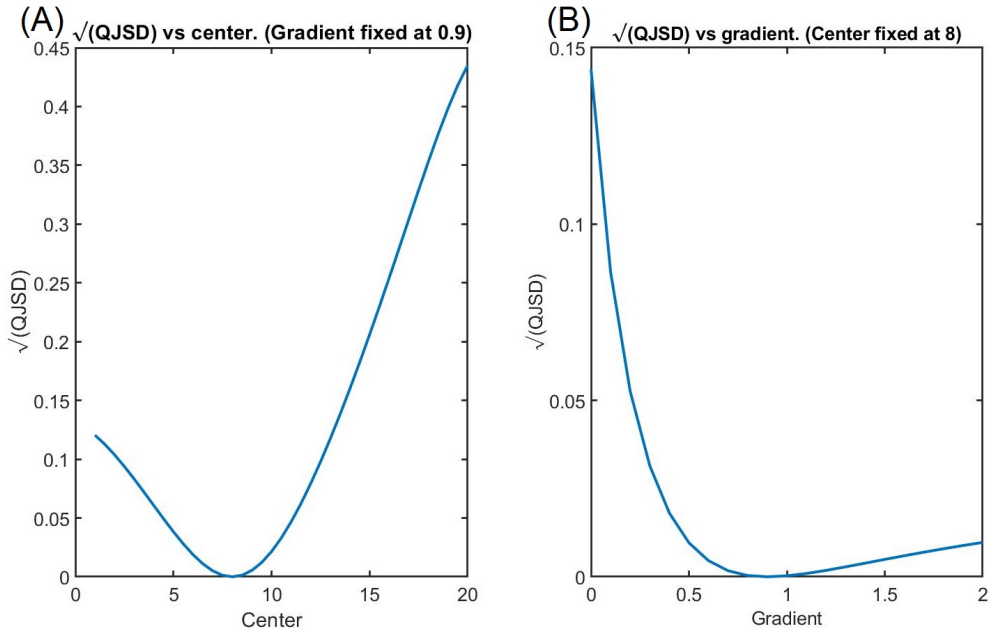

FIGURE 3.11: Using our sigmoid model (Eq.3.10) and some additional assumptions this figure show the $\sqrt{JS_{\mathcal{N}}}$ between a fixed form ($a = 0.9, c = 8$) and a form where we vary one variable at a time (a or c). (A) Shows the effect on the distance when we fix the gradient and vary the location of the centre. (B) Shows the effect on the distance when we fix the centre and vary the value of the gradient.

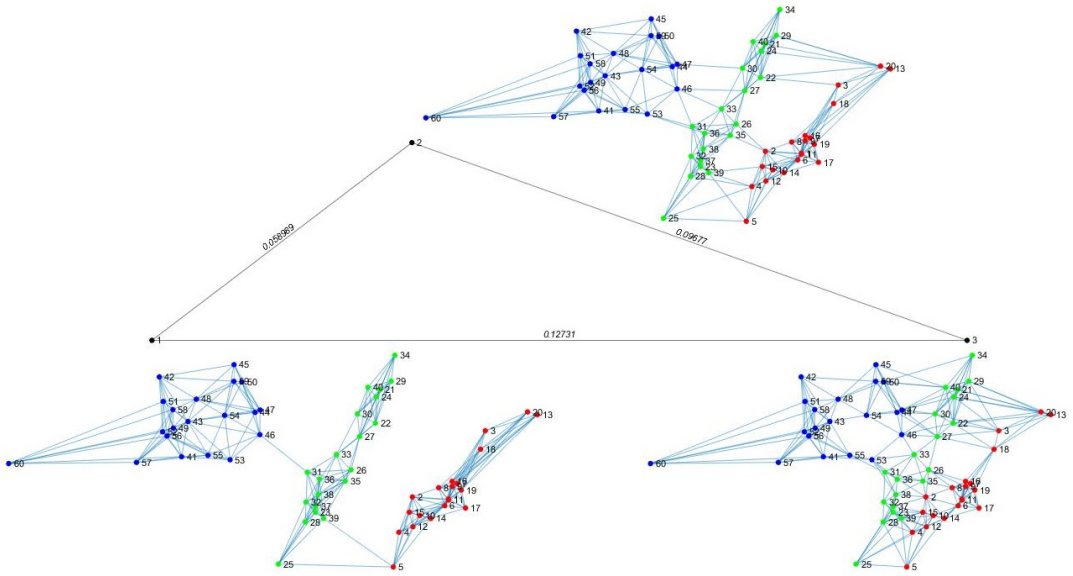

FIGURE 3.12: Three clusters of points and the three KNN networks that arise as the clusters are moved closer together. Network 1 has an entropy of 5.7311, network 2 with 5.742 and network with 5.7469. The central triangle is and embedding of these three networks where the edges are equal to the $\sqrt{JS_{\mathcal{N}}}$ between these networks.

Figure 3.12. Here, we have generated three clusters, of 20 points, from different bivariate distributions, giving them a geometric embedding in the plane. Then we have formed a KNN network with $k = 7$ and colored the nodes corresponding to each cluster. This time, as opposed to simply randomly rewiring the network, we moved the outer clusters of point, red and blue, in towards the middle cluster and then reformed the KNN network. Initially, network 1 in Figure 3.12, these were centered at $(-2, 3)$, $(0, 2)$ and $(2, 1)$ and the outer two clusters were then moved at increments of 0.5, yielding networks 2 and 3. In the center of Figure 3.12 is a triangle with edges equal to the $\sqrt{JS_{\mathcal{N}}}$ between the networks.

In the first network, network 1, the clustering of the separate distributions are clear and distinct, with only 3 edges in total connecting the clusters whose entropy is 5.7311. The second network, after one increment, has a very similar structure to the first. Whilst the number of edges between the clusters have increased, to 19, the separation of the distributions is fairly clear and would mostly achieve the same clustering. This similarity is reflected in their distance of 0.058989 how ever the increase in disorder is reflected in its entropy rising to 5.742. Now compare the first network to the third. The third network has had the outer clusters moved in by two increments and the resulting KNN network formed has completely change. The distributions are no longer clear, its entropy has risen to 5.7469, and if clustered would result a clustering different from that of the other two networks. This difference between the first and third network can be seen with the distance increasing up to 0.12731. This experiment was repeated 100 times, each

time using a different set of distributions to generate the points. In every single instance the original network was closer to the second network than the third, with respect to $\sqrt{JS_{\mathcal{N}}}$, showing how $\sqrt{JS_{\mathcal{N}}}$ captures the structural similarities between the networks.

Chapter 4

Entropy Network Fusion

This Chapter introduces our entropy-based data integration methodology which we call Entropy Network Fusion (ENF). First we discuss the motivation behind ENF, followed by a summary of the methodology. Then we take an in-depth look at the methodology, starting with a discussion of the input data, the cost function, an adaptation for a weighted version, followed by the derivation of the gradient for the cost function and an numerical check that the gradient is correct. This Chapter concludes by studying the convergence of our method. We are able to show that our cost function is both bounded and convex, it produces sensible non-trivial answers with a very small variance in the output when inititalised from different random points and that our derivation for the gradient is correct.

4.1 Motivation

In Section 1.1 we explained how each omic data set probes a different aspect of the same underlying biological system. For example transcriptomics captures information about RNA transcripts and the active components of the cell; proteomics focuses on the identification and quantity of proteins within the cell [16]. Each of these datasets reflects a particular aspect of the same complex biological dynamical system, and a more complete understanding will not arise from any single data set [7–9]. Therefore it makes sense to combine, or integrate, the information on these data sets, as they all share the same underlying 'ground truth', that is, each data set arises from the same underlying biological system.

There are many methods available, as discussed in Section 1.1, for the integration of omic datasets. Whilst all have to overcome the same set of challenges (dimensionality, noise, concordant and discordant information [1, 12, 14, 17]), all of them have their own sets of advantages, disadvantages and restrictions. Different methods may have distinct

objectives that they set out to achieve and may be designed for a specific data integration task. Therefore, the choice of integration tool will depend on the objective of the analysis. In the case of this project, the objective is to cluster patient data into clinically-consistent groups based on the integration of a more comprehensive set of data, identifying potentially novel subgroups and new features of these subgroups. We do this by encoding different omics data into networks, and then integrating, or fusing, all networks into a single one, from which a clustering solution is obtained.

As previously discussed, representing these data sets as networks is a popular strategy and widely used in the analysis of biological systems [1, 22, 24]. Representing the data as networks not only helps with visualisation but also reveals structure that may not, at first, be apparent. Many of these structural properties are encoded in the eigenvalues and eigenvectors of matrices associated to the network, such as the graph laplacian. For example, the multiplicity of the 0 eigenvalue equals the number of connected components of the network [68, 69]; the eigenvalues are used to estimate the number of clusters and the eigenvectors to find clusters (spectral clustering [53, 54]); they can also be used to optimally embed the network into n dimensions (spectral embedding [54, 70]). Because the goal of the project is to cluster patients, a network representation is appropriate. Hence the result of any integration method should, in terms of structure, correspond to the structure of the various sources used as close as possible. Therefore, using the structural information during the integration process should be a key feature, and a good proxy for this are the eigenvalues of the laplacian.

The network entropy is defined in terms of the (normalised) laplacian eigenvalues, (Eq 3.6 and [13]), and these eigenvalues are related to the structural properties of the network. As reviewed in Section 1.2, entropy has been used to quantify the complexity of networks and the distance/dissimilarity between networks [38–44]. This was then further investigated and demonstrated in Chapter 3 showing how the structural differences between networks is quantified by the QJSD. Therefore we shall use network entropy and network QJSD to develop a network integration methodology which shall seek a single network that is as close as possible, in terms of structure, to the given input networks. We call this method Entropy Network Fusion (ENF).

Although this method is motivated by patient omic data, it can be applied to any general network integration task that satisfies the same conditions/objectives. That is, if given a set of networks on the same set of objects and the underlying true clustering structure is being sought, then, ENF can accomplish this task.

4.2 Input

The inputs for ENF are l matrices of size $n \times n$ that represent (as adjacency matrices) l similarity networks on the same set of n objects (in our case, n patients and l types of omics data). Each of the similarity networks quantify how similar the set of patients are with respect to a different data type. There are only two conditions that are enforced upon the inputs. Firstly, the similarites of the network are bounded (normalised) between zero and one, such that if two patients are identical (in our network model) then they have similarity equal to one, and zero for when they are completely dissimilar. As standard in network clustering, the edge weights represent the cost of removing an edge, hence the more similar two patients are, the higher the cost of assigning them to different clusters. Secondly, the similarity networks must be undirected, as similarity is naturally a symmetric relation. This means that the resulting matrices are symmetric and, in particular, they have real eigenvalues and therefore the entropy can be calculated.

To allow for maximal range of application, ENF does not construct the similarity networks, leaving that to the user to create them how they wish. This is because the user may have their own preferred method, the data they are working with may require the similarities to be constructed in a particular fashion or they may even wish to have a different method for each layer they construct. Regardless of how the user constructs the network they must however satisfy the following requirements for ENF to be used: Non-negative entries, that is, none of the edge weights can be negative; Symmetric, that is, the edges can not be directed so that the similarity of object i to object j is the same as the similarity of object j to object i; The entries are normalised such that 1 indicates similarity and 0 indicates non-similarity. Note that there has been recent work extending the entropy to directed networks [79], whose extension to QJSD and this method, could be explored in future work.

To conclude this subsection, we cover a simple method a user can use to construct similarity networks from the raw data as follows, after having normalised each column of the data. First, for each data type, a euclidean distance matrix D should be created between the n patients resulting in a $n \times n$ matrix. Then each of these matrices are transformed according to:

$$S_{ij} = \exp\left(\frac{-D_{ij}}{\mu}\right). \tag{4.1}$$

Here μ is a scalar to control the drop off of the exponential, and will be referred to as the *scale factor*. The scale factor is used to deal with the scaling problem that arises from normalised high dimensional data, in that two points will typically have a larger distance to each other in a higher dimensional vector space compared to their distance in a lower dimensional space. If there is a large variance in the number of columns for the different date sets there will be a corresponding variance in the distances matrices. This can be referred to as the 'Curse of Dimensionality', hence the need for scaling of some

form. For ENF, μ, is set equal to the average of unique pairwise distance between points in that given layer. This can be thought of transforming the distances to be percentages above the average distance in the given layer and therefore brings different layers onto the same scale. This transforms the distance matrices into similarity matrices. Equation 4.1 satisfies the requirement $0 \leq S_{ij} \leq 1$. Two patients, identical in every feature, have a distance equal to zero and hence have similarity equal to one. The distance between two patients will increase as they become more dissimilar, causing their similarity tending towards zero. The symmetric requirement is fulfilled as euclidean distance is a symmetric relationship and is preserved by our transformation.

4.3 Cost Function

As previously stated our objective is to find the network that is structurally closest to all of the given input networks. This is an optimisation problem. These types of problems occur everywhere, from investment portfolios (maximising return whilst minimising risk) to optimum warehouse locations for delivery companies [80]. To solve these types of problems, we need a way of measuring the goodness of a solution, namely a cost (or objective) function. This function is then minimised or maximised, depending on the setup, to find the 'best' solution [80]. The cost function used here is based upon the average of squared errors, similar to that used in linear regression [77, 78], and the error is the distance between the networks as the square root of $\sqrt{JS_{\mathcal{N}}}$. Recall that $JS_{\mathcal{N}}$ is defined as:

$$JS_{\mathcal{N}}(G_1, G_2) = H_{\mathcal{N}}(g_2 G_1 + g_1 G_2) - \frac{1}{2}(H_{\mathcal{N}}(G_1) + H_{\mathcal{N}}(G_2))$$

As we saw in Chapter 3.4 $JS_{\mathcal{N}}$ quantifies the difference in the clustering structure between two networks by using the information contained in their eigenvalues. By using this in our cost function our output shall have a structure that is as close as possible to the structure of all the input networks, identifying structure that is common to all and not being corrupted by conflicting information. We use θ to denote the adjacency matrix of the network that is our current solution, when this is put into the cost function the lower the resulting cost, the better the current solution is. Therefore the cost function for ENF, C, at a given solution θ, is defined as:

$$C(\{S^{(l)}\}, \theta) = \frac{1}{l} \sum_{m=1}^{l} \sqrt{JS_{\mathcal{N}}(S^{(m)}, \theta)}^2 = \frac{1}{l} \sum_{m=1}^{l} JS_{\mathcal{N}}(S^{(m)}, \theta)$$

We can then rewrite the cost function in terms of the entropy of the networks only, saving computational time, as

$$C(\{S^{(l)}\},\theta) = \frac{1}{l}\sum_{m=1}^{l}\left[H_{\mathcal{N}}\left(tS^{(m)} + s^{(m)}\theta\right)\right] - \underbrace{\frac{1}{2l}\sum_{m=1}^{l}[H_{\mathcal{N}}(S^{(m)})]}_{constant} - \frac{1}{2}H_{\mathcal{N}}(\theta) \tag{4.2}$$

where $t = \sum_{ij}\theta_{ij}$ and $s^{(m)} = \sum_{ij} s^{(m)}_{ij}$. By rewriting it in this form, the constant $\sum_{m=1}^{l}[H_{\mathcal{N}}(S^{(m)})]$, needs only to be calculated once at the very start of the process and the entropy of θ is only needed to be calculated once per iteration. Instead of being recalculated every iteration and distance.

Here, l is the number of layers (or datasets) being integrated; $S^{(m)}$ refers to the mth similarity network, as used in Section 2.1 and Chapter 2.2; $H_{\mathcal{N}}(\cdot)$ is the network entropy as defined in Eq. 3.6. At the start of the process θ is initialised to be a random symmetric matrix of order $n \times n$, n being the number of patients, with each entry between 0 and 1. In practice we found that setting θ equal to the average of the given similarity matrices greatly accelerated the convergence of ENF, which will be demonstrated in Section 4.8. However if any of the given similarity matrices are on significantly different scale then doing this will result in that layer dominating θ and in fact slowing down the convergence.

The cost function defined above also comes with the added properties of being bounded and scale independent. The bounded property comes from the fact that the square root of QJSD is bounded between 0 and 1 [61–63],

$$\begin{aligned} 0 \leq \sqrt{JS_{\mathcal{N}}(S^{(m)},\theta)} &\leq 1 \\ \Rightarrow 0 \leq JS_{\mathcal{N}}(S^{(m)},\theta) &\leq 1 \\ \Rightarrow 0 \leq \frac{1}{l}\sum_{m=1}^{l} JS_{\mathcal{N}}(S^{(m)},\theta) &\leq 1 \end{aligned}$$

As for the scale independence property this can be seen from a closer inspection of the QJSD. Consider a real symmetric matrix A, then the eigenvalues of αA are the same as those of A scaled by α, becoming $\alpha\lambda_i$. When these eigenvalues are normalised, in the calculation of the entropy, they become the same value:

$$\begin{aligned} \frac{\alpha\lambda_i}{\sum_{k=1}^{n}(\alpha\lambda_k)} &= \frac{\alpha\lambda_i}{\alpha\sum_{k=1}^{n}(\lambda_k)} \\ &= \frac{\lambda_i}{\sum_{k=1}^{n}(\lambda_k)}. \end{aligned}$$

Hence the entropy is equal, $H_{\mathcal{N}}(\alpha A) = H_{\mathcal{N}}(A)$. Then upon considering QJSD it can be seen:

$$\begin{aligned} JS_{\mathcal{N}}(\alpha A, \beta B) &= H_{\mathcal{N}}\Big((\beta b)(\alpha A) + (\alpha a)(\beta B)\Big) - \frac{1}{2}(H_{\mathcal{N}}(\alpha A) + H_{\mathcal{N}}(\beta B)) \\ &= H_{\mathcal{N}}\Big((\alpha\beta)(bA + aB)\Big) - \frac{1}{2}(H_{\mathcal{N}}(A) + H_{\mathcal{N}}(B)) \\ &= H_{\mathcal{N}}(bA + aB) - \frac{1}{2}(H_{\mathcal{N}}(A) + H_{\mathcal{N}}(B)) \\ &= JS_{\mathcal{N}}(A, B). \end{aligned}$$

(Recall that a and b are the sum of the entries of A and B, respectively.) This means that any scalar multiple of the given output will also achieve the same cost and performance.

Whilst in our main formulation of the ENF cost function the distance to each layer is weighted equally, it can be easily extended to including weights for the distance to each layer. This situation could arise if the user possess prior knowledge and wishes to prioritize a particular layer in the fusion. The weighted ENF (wENF) cost function is given by:

$$C(\{S^{(l)}, w_l\}, \theta) = \frac{1}{\sum_{m=1}^{l} w_m^2} \sum_{m=1}^{l} \left(w_m \sqrt{JS_{\mathcal{N}}(S^{(m)}, \theta)} \right)^2 \tag{4.3}$$

$$= \frac{1}{\sum_{m=1}^{l} w_m^2} \sum_{m=1}^{l} \left[w_m^2 H_{\mathcal{N}} \left(tS^{(m)} + s^{(m)}\theta \right) \right] \tag{4.4}$$

$$- \frac{1}{2\sum_{m=1}^{l} w_m^2} \sum_{m=1}^{l} w_m^2 [H_{\mathcal{N}}(S^{(m)})] - \frac{1}{2} H_{\mathcal{N}}(\theta)$$

where $w_1, \ldots, w_l$ are the corresponding weights for each layer. However in the analysis performed in Section 5.2.2 we only explore the unweighted case.

Note that, in the ENF cost function, we explicitly compute the eigenvalues, and this becomes the most costly computational step. However if the matrix formulation, (Eq. 3.3), were used in the cost function instead, this could give a speed-accuracy trade off. The loss of accuracy results from the approximate calculation of $\log(\rho)$, which for a matrix is given by an infinite sum. On the other hand, the computational speed gain comes from avoiding the full eigenvalue computation and taking matrix products instead. This is something that could be explored in future work

4.4 Gradient

With our cost function formed for this problem we now need to optimise it via some algorithm. A popular way to minimise these functions is to use the derivative of the cost function. By moving in the direction of the gradient, it will decrease the cost most rapidly [80]. Hence the next step is to calculate the derivative. This requires differentiating our cost function, to see how the entropy changes with respect to the eigenvalues. The smoothness of our cost function follows from the continuity and differentiability of the eigenvalues [81] and of the composition of smooth functions. Once we have the gradient the cost function can then be minimised by gradient descent, or any other method that the user wishes to implement. Note that in the following subsection some definitions are repeated for completeness. As $JS_{\mathcal{N}}$ is defined as $H_{\mathcal{N}}(aG + gA) - \frac{1}{2}(H_{\mathcal{N}}(G) + H_{\mathcal{N}}(A))$, the computation of the gradient is divided into two parts, first $H_{\mathcal{N}}(G)$ followed by $H_{\mathcal{N}}(aG + gA)$.

4.4.1 Derivative of Eigenvalues

As network entropy is defined in terms of the eigenvalues of a matrix, first, we cover the first derivative of eigenvalues. We use a perturbation theory result (Theorem 6.3.12, from [71]) to this end. Suppose that we have a simple eigenvalue λ of a matrix M, with left and right eigenvectors y^T and x. If M is subjected to a change E, then there is a unique eigenvalue $\lambda(t)$ of $M + tE$. This function, $\lambda(t)$, is both continuous and differentiable at 0, with $\lim_{t\to 0} \lambda(t) = \lambda$. But more importantly, the derivative at $t = 0$ is given by:

$$\frac{d\lambda}{dt} = \frac{y^T E x}{y^T x}. \tag{4.5}$$

Later we shall be abusing this notation slightly by using $\frac{d\lambda}{dE}$ to represent the derivative, this is because the particular change occurring in the matrix will be important. We shall use $E(i, j)$ to represent the *elementary change* in the (i, j)-th and (j, i)-th entry of a adjacency matrix. Formally, this is another matrix that is defined as:

$$E(k, f)_{ij} := \begin{cases} 1 & (k = i, f = j) \text{ or } (f = i, k = j), \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

This maintains the symmetric property of the matrix and of the underlying graph it represents. Note for the graph it represents, the addition of the elementary change to the adjacency matrix is just changing the value of one particular edge weight (k, f). Recall from Chapter 3 that the entropy of a graph is defined in terms of the eigenvalues of its laplacian,

$$H_{\mathcal{N}}(G) = -\sum_{\lambda_i \in \sigma(L(G))} \frac{\lambda_i}{\sum_j \lambda_j} \log_2 \left(\frac{\lambda_i}{\sum_j \lambda_j} \right) \tag{4.7}$$

Proceeding forward we use λ_i to denote the i-th eigenvalue of the matrix, y_i and x_i to denote the left and right eigenvectors associated with λ_i, whose norms equal to one and y_{ji} to denote the j-th entry of i-th vector. As we are dealing with real symmetric matrices the left and right eigenvectors are in fact equal ($y^T = x$).

Given a change in the original matrix $G' = G + E(k, f)$, the new Laplacian is $L(G') = L(G) + L(E(k, f))$, since the Laplacian is linear. Therefore the change that is to be used in the calculation of the derivative for the eigenvalues is $L(E(k, f)) = E_{\mathcal{L}}(k, f)$. To be explicit, it is defined as:

$$E_{\mathcal{L}}(k,f)_{ij} = L(E(k,f))_{ij} = \begin{cases} 1 & i = j = k, f \\ -1 & (k = i, f = j) \text{ or } (f = i, k = j) \\ 0 & \text{otherwise} \end{cases}$$

For the function $H_{\mathcal{N}}(G)$, the change in the laplacian is the one described above, $E_{\mathcal{L}}(k, f)$. Therefore the derivative of its eigenvalues, λ_i, with respect to an elementary change is given by using $E_{\mathcal{L}}(k, f)$ in Eq 4.5.

$$\begin{aligned} \frac{\partial \lambda_i}{\partial G_{kf}} = \frac{d\lambda}{dE_{\mathcal{L}}(k,f)} &= x_i^T E_{\mathcal{L}}(k,f) x_i \\ &= \sum_a \sum_b x_{ai} E_{\mathcal{L}}(k,f)_{ab} x_{bi} \\ &= x_{ki}^2 - 2x_{ki}x_{fi} + x_{fi}^2 \\ &= (x_{ki} - x_{fi})^2 \end{aligned} \tag{4.8}$$

Here we are abusing notation, by using $\frac{\partial}{\partial G_{ij}}$ to mean the derivative with respect to the G_{ij} *and* the G_{ji} entry as both entries are being change to preserve the symmetric property of the graph.

Note that if the two entries k and f are fixed, not equal, and this term is summed over all the eigenvectors, indexed by i, then the sum is equal to 2, otherwise it is equal to 0,

$$\begin{aligned} \sum_{i=1}^{n} (x_{ki} - x_{fi})^2 = ||y_k - y_f||^2 \\ &= \langle y_k - y_f, y_k - y_f \rangle \\ &= \langle y_k, y_k \rangle - 2\langle y_f, y_k \rangle + \langle y_f, y_f \rangle \\ &= 1 + 0 + 1 = 2 \end{aligned} \tag{4.9}$$

This is because the sum is equal to the norm of the difference between the k-th and f-th left eigenvector. The final line follows from the eigenvectors having norm equal to 1 and eigenvectors corresponding to different eigenvalues are necessarily orthogonal. This simplification will be used later on.

Next we consider the function $H_{\mathcal{N}}(aG + gA)$, where A is being kept fixed and the derivative is with respect to the entries of G. If G is subjected to an elementary change in the (i, j)-entry, $i \neq j$, the resulting change that occurs in the laplacian can be seen in the following:

$$\begin{aligned} L(a(G + E(i, j)) + (g + 2)A) &= L((aG + gA) + (aE(i, j) + 2A)) \\ &= L(aG + gA) + L(aE(i, j) + 2A) \\ &= L(aG + gA) + aL(E(i, j)) + 2L(A), \end{aligned}$$

where $a = \sum_{ij} A_{ij}$ and $g = \sum_{ij} G_{ij}$. (For a formal proof, see Appendix A.) Therefore the change, in this case, that is used for the derivative of the eigenvalues is $aE_{\mathcal{L}}(i, j)+2L(A)$. Using this in Eq 4.5 gives the derivative of the eigenvalues of the laplacian:

$$\begin{aligned} \frac{\partial \lambda_i}{\partial G_{kf}} &= x_i^T (aE_{\mathcal{L}}(k, f) + 2L(A))x_i \\ &= a(x_{ki} - x_{fi})^2 + 2x_i^T L(A)x_i \end{aligned} \tag{4.10}$$

4.4.2 Part 1: $H_{\mathcal{N}}(G)$

Now that we have covered the preliminaries we now move onto calculating both the first and second derivatives of both the necessary cases, beginning with $H_{\mathcal{N}}(G)$. To differentiate $H_{\mathcal{N}}(G)$ with respect to the entries of the adjacency matrix of G, a change of variables is needed as the entropy is defined in terms of the eigenvalues of the laplacian.

$$\frac{\partial H_{\mathcal{N}}(G)}{\partial G_{ij}} = \sum_{k=1}^{n} \frac{\partial H_{\mathcal{N}}(G)}{\partial \lambda_k} \frac{\partial \lambda_k}{\partial G_{ij}} \tag{4.11}$$

In the previous subsections we calculated the derivative of $\frac{\partial \lambda_k}{\partial G_{ij}}$ for this case, Eq 4.8. Next we compute $\frac{\partial H_{\mathcal{N}}(G)}{\partial \lambda_k}$, differentiating entropy (Eq. 4.7) with respect to the λ_i. To do this, the computation is divided into several stages for ease:

$$\frac{\partial}{\partial \lambda_i}\left(\frac{\lambda_k}{\sum_j \lambda_j}\right) = \begin{cases} \frac{\sum_{j\neq i} \lambda_j}{(\sum_j \lambda_j)^2} & k = i, \\ \frac{-\lambda_k}{(\sum_j \lambda_j)^2} & \text{otherwise.} \end{cases} \tag{4.12}$$

$$\frac{\partial}{\partial \lambda_i}\left(\log_2\left(\frac{\lambda_k}{\sum_j \lambda_j}\right)\right) = \begin{cases} \frac{\sum_{j\neq i} \lambda_j}{\log(2)\lambda_i \sum_j \lambda_j} & k = i, \\ \frac{-1}{\log(2)\sum_j \lambda_j} & \text{otherwise.} \end{cases} \tag{4.13}$$

Combine these two to differentiate one term of Eq. 4.7, where $k = i$:

$$\frac{\partial}{\partial \lambda_i}\left(\frac{\lambda_i}{\sum_j \lambda_j}\log_2\left(\frac{\lambda_i}{\sum_j \lambda_j}\right)\right) = \frac{\sum_{j\neq i} \lambda_j}{\log(2)(\sum_j \lambda_j)^2} + \frac{\sum_{j\neq i} \lambda_j}{(\sum_j \lambda_j)^2}\log_2\left(\frac{\lambda_i}{\sum_j \lambda_j}\right) \tag{4.14}$$

For any other k:

$$\frac{\partial}{\partial \lambda_i}\left(\frac{\lambda_k}{\sum_j \lambda_j}\log_2\left(\frac{\lambda_k}{\sum_j \lambda_j}\right)\right) = \frac{-\lambda_k}{\log(2)(\sum_j \lambda_j)^2} - \frac{\lambda_k}{(\sum_j \lambda_j)^2}\log_2\left(\frac{\lambda_k}{\sum_j \lambda_j}\right) \tag{4.15}$$

Therefore combining these together and setting $T = \sum_j \lambda_j = Tr(L)$:

$$\begin{aligned}
\frac{\partial H_{\mathcal{N}}(G)}{\partial \lambda_i} &= \frac{-1}{T^2}\left[\frac{\sum_{j\neq i}\lambda_j}{\log(2)} + (\sum_{j\neq i}\lambda_j)\log_2\left(\frac{\lambda_i}{T}\right) + \sum_{k\neq i}\left[\frac{-\lambda_k}{\log(2)} - \lambda_k\log_2\left(\frac{\lambda_k}{T}\right)\right]\right] \\
&= \frac{-1}{T^2}\left[(\sum_{j\neq i}\lambda_j)\log_2\left(\frac{\lambda_i}{T}\right) - \sum_{k\neq i}\left[\lambda_k\log_2\left(\frac{\lambda_k}{T}\right)\right]\right] \\
&= \frac{-1}{T^2}\left[(\sum_{j\neq i}\lambda_j)\log_2\left(\frac{\lambda_i}{T}\right) - \sum_{k\neq i}\left[\lambda_k\log_2\left(\frac{\lambda_k}{T}\right)\right] + \lambda_i\log_2\left(\frac{\lambda_i}{T}\right) - \lambda_i\log_2\left(\frac{\lambda_i}{T}\right)\right] \\
&= \frac{-1}{T^2}\left[(\sum_{j}\lambda_j)\log_2\left(\frac{\lambda_i}{T}\right) - \sum_{k}\left[\lambda_k\log_2\left(\frac{\lambda_k}{T}\right)\right]\right] \\
&= \frac{-1}{T}\left[\log_2\left(\frac{\lambda_i}{T}\right) + H_{\mathcal{N}}(G)\right]
\end{aligned} \tag{4.16}$$

Hence we have how the entropy changes with respect to the eigenvalues. Finally combining (Eq 4.16) and (Eq 4.8) into (Eq4.11), the equation for the derivative of entropy with respect to a particular entry is given by:

$$\begin{aligned}
\frac{\partial H_{\mathcal{N}}(G)}{\partial G_{ij}} &= \frac{-1}{T}\sum_{k=1}^{n}\left[\log_2\left(\frac{\lambda_k}{T}\right) + H_{\mathcal{N}}(G)\right](x_{ik} - x_{jk})^2 \\
&= \frac{-1}{T}\sum_{k=1}^{n}\left[(x_{ik} - x_{jk})^2\log_2(\lambda_k)\right] + \frac{2}{T}\log_2(T) - \frac{2}{T}H_{\mathcal{N}}(G)
\end{aligned} \tag{4.17}$$

If $i = j$ then $\frac{\partial H_{\mathcal{N}}(G)}{\partial G_{ij}} = 0$ and the last step is made by using (Eq 4.9).

4.4.3 Part 2: $H_{\mathcal{N}}(aG + gA)$

For this second case, the only difference between this and the first case is the change that occurs in the laplacian and its corresponding effects on the derivative of the eigenvalues. This was calculated in Subsection 4.4.1, specifically $aE_{\mathcal{L}}(i,j)+2L(A)$, where $a = \sum_{i,j} A$. The chain rule must still be applied, giving the same two components that we had in the first case:

$$\frac{\partial H_{\mathcal{N}}(aG + gA)}{\partial G_{ij}} = \sum_{k=1}^{n}\frac{\partial H_{\mathcal{N}}(aG + gA)}{\partial \lambda_k}\frac{\partial \lambda_k}{\partial G_{ij}}$$

This first component, $\frac{\partial H_{\mathcal{N}}(aG+gA)}{\partial \lambda_k}$ is exactly the same as the first case, resulting in Eq. 4.16. Combining this, as above, with the appropriate derivative of eigenvalues (Eq 4.10) we arrive at:

$$\frac{\partial H_{\mathcal{N}}(aG+gA)}{\partial G_{ij}} = \frac{-1}{T}\sum_{k=1}^{n}\left[\log_2\left(\frac{\lambda_k}{T}\right) + H_{\mathcal{N}}(aG+gA)\right]\left[a(x_{ik}-x_{jk})^2 + 2x_k^T L(A)x_k\right] \tag{4.18}$$

$$= \frac{-1}{T}\sum_{k=1}^{n}\log_2(\lambda_k)\left[a(x_{ik}-x_{jk})^2 + 2x_k^T L(A)x_k\right] + \frac{2}{T}\left[\log_2(T) - H(aG+gA)\right]\left[a + \sum_{k=1}^{n} x_k^T L(A)x_k\right] \tag{4.19}$$

4.4.4 ENF Gradient

Finally, as differentiation is linear, we can combine the derivatives above to get the derivatives of our ENF cost function Eq 4.2. For example, the first derivative is given by:

$$\begin{aligned}\frac{\partial C(\{G^{(l)}\},\theta)}{\partial \theta_{ij}} &= \frac{1}{l}\sum_{m=1}^{l}\left[\frac{\partial H_{\mathcal{N}}\left(tG^{(m)}+g^{(l)}\theta\right)}{\partial \theta_{ij}}\right] - \frac{1}{2}\frac{\partial H_{\mathcal{N}}(\theta)}{\partial \theta_{ij}} \\ &= \frac{-1}{l}\sum_{m=1}^{l}\left[\frac{1}{T^{(m)}}\sum_{k=1}^{n}\left[\log_2\left(\frac{\lambda_k^{(m)}}{T^{(m)}}\right) + H_{\mathcal{N}}(tG^{(m)}+g^{(m)}\theta)\right]\cdot \right. \\ &\qquad \left.\cdot\left[g^{(m)}(x_{ik}^{(m)}-x_{jk}^{(m)})^2 + 2x_k^{(m)T}L(G^{(m)})x_k^{(m)}\right]\right] \\ &\qquad + \frac{1}{2U}\sum_{k=1}^{n}\left[\log_2\left(\frac{\epsilon_k}{U}\right) + H_{\mathcal{N}}(\theta)\right](\mathcal{X}_{ik}-\mathcal{X}_{jk})^2,\end{aligned} \tag{4.20}$$

where $\lambda^{(m)} \in \sigma(L(tG^{(m)}+g^{(m)}\theta))$, $x_k^{(m)}$ is the right eigenvector associated with $\lambda_k^{(m)}$, $T^{(m)} = \sum_j \lambda_j^{(m)}$, $\epsilon \in \sigma(L(\theta))$, $\mathcal{X}_k$ is the eigenvector associated to ϵ_k and $U = \sum_j \epsilon_j$. The corresponding second derivative is given by:

$$\frac{\partial C(G^{(l)},\theta)}{\partial \theta_{\alpha\beta}\partial \theta_{ij}} = \frac{1}{l}\sum_{m=1}^{l}\left[\frac{\partial H_{\mathcal{N}}\left(tG^{(m)}+g^{(l)}\theta\right)}{\partial \theta_{\alpha\beta}\partial \theta_{ij}}\right] - \frac{1}{2}\frac{\partial H_{\mathcal{N}}(\theta)}{\partial \theta_{\alpha\beta}\partial \theta_{ij}}. \tag{4.21}$$

The derivation of Eq 4.21 is given in Appendix C and can be used in second order optimisation methods, such as Newtons method [80, 82]. However, this is very costly in terms of computational power due to the vast number of terms that needs to be calculated. If the network has n vertices then there are $\frac{n(n-1)}{2}$ variables hence $\frac{n^2(n-1)^2}{4}$ partial deriatives to compute. Therefore we proceed using only first order methods to minimise the cost function.

We can numerically validate the correctness of our 1^{st} derivative formula by comparing it to the approximation given by Finite Difference (FD). Finite difference approximates the gradient by making a small adjustment in a variable and looking at corresponding change in the function. More specifically, the one-sided difference is given by the formula:

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x + \epsilon e_i) - f(x)}{\epsilon}$$

where ϵ is the size of the adjustment and e_i is unit basis of the entry being adjusted [80]. Figure 4.1 shows the comparison of Eq. 4.20 against FD when using a random graph, as one of our input layers which remains fixed, and a tree graph for our θ , whose entries the derivatives are being take with respect to. As we can see from Figure 4.1 D and E, the structure of the two gradients are completely identical, including being on the same numerical scale. This is confirmed further upon looking at Figure 4.1 F and noting that the absolute difference of these two gradients is on the order of 10^{-6}. This type of analysis was repeated for other graphs and on the separate components of the gradient, $\frac{\partial H_{\mathcal{N}} G}{\partial G_{ij}}$ and $\frac{\partial H_{\mathcal{N}}(aG+gA)}{\partial G_{ij}}$, achieving the same results.

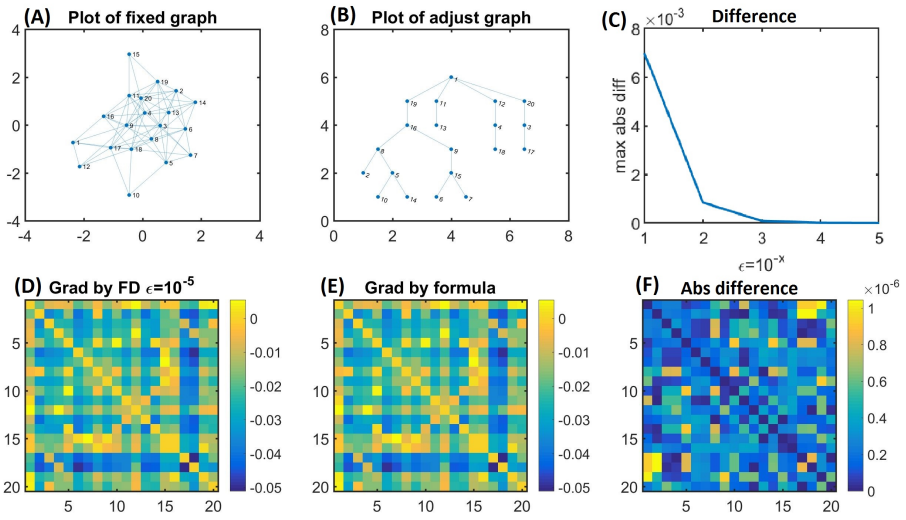

FIGURE 4.1: Shown here are the results from the numerical comparison between the FD estimation of the gradient and the formula derived in Eq. 4.20. (A) Is the graph that is being kept fixed, the term G. (B) The graph that is being adjusted and the gradient being calculated with respect to, θ. (C) The maximum absolute difference between Eq. 4.20 and FD as 10^{-x} decreases (the x-axis being the value for x). (D) The estimated gradient produced by FD using $\epsilon = 10^{-5}$. (E) The gradient produced using Eq. 4.20. The axes for these last two sub-figures correspond to the matrix entry. (F) The absolute difference between (D) and (E)

4.4.5 Minimisation

As explained at the beginning of this subsection, we shall be using the 1^{st} derivative to minimise the cost function (Eq 4.2). There are multiple first order optimisation procedures for minimising a function, so called because they use the first derivative of the function. A selection of these methods are covered in [80, 82]. The most common is *Batch Gradient Descent* or simply *Gradient Descent*. Here, for each iteration, the entries of θ_{ij} are adjusted according to:

$$\theta_{ij} - \alpha \frac{\partial C(\{G^{(l)}\}, \theta)}{\partial \theta_{ij}}. \tag{4.22}$$

Here α is a learning rate variable that is set to control the descent. For this situation after this adjustment is made, if any of the θ_{ij} are less than 0 then they are set equal to 0. This is as θ is a graph with edge weights restricted to be non-negative. This is then repeated for a given number of iterations.

Whilst gradient descent is the simplest method to implement, other methods are known to converge faster. One such method consists of adding a *momentum* term [80, 82]. In essence the momentum term accumulates past gradients: when they agree they increase the corresponding change and when they dissagree they reduce the change. The solution is adjusted according to:

$$\begin{aligned} \nu_t &= \gamma \nu_{t-1} + \alpha \frac{\partial C(\{G^{(l)}\}, \theta_t)}{\partial \theta_t} \\ \theta_{t+1} &= \theta_t - \nu_t \end{aligned} \tag{4.23}$$

where γ is a scalar in the range $(0, 1)$, used to decay the past gradients, making recent gradients more important than earlier ones.

There are three other methods that we wish to mention but we shall not be covering in detail; they can be found in [80, 82]. The first is *Nesterov accelerated gradient* (NAG). This looks ahead to incorporate the gradient of an approximation of the next iteration, so the parameters can be updated more carefully. The drawback with this method is that the computational time is doubled due to calculating another gradient. The next two methods are similar in their motivation behind their design, these are *Adagrad* and *Adadelta*. Both of these adapt their learning rates for each entry based on their importance, rather than the adjustments being uniform. Where they differ is how they adjust their learning rates. Adagrad uses the past gradients the determine how the learning rate should be changed whilst Adadelta uses the previous updates and gradients to determine its learning rate. In fact, Adadelta eliminates the learning rate entirely, leaving it to figure out its own learning rate.

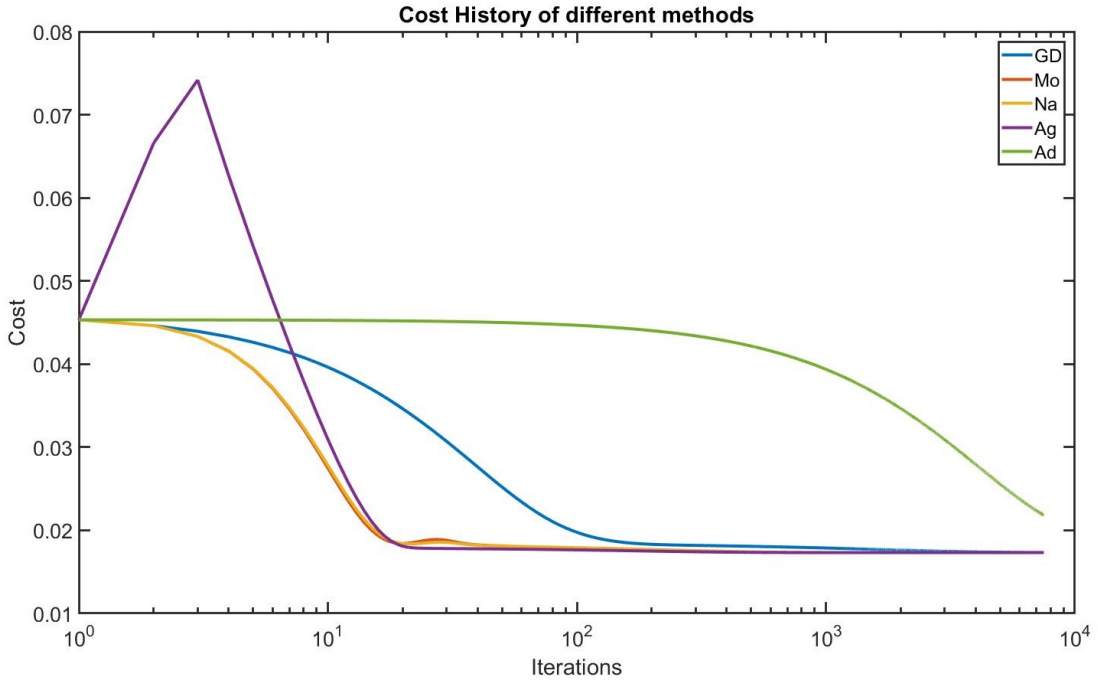

FIGURE 4.2: This figure shows the difference in cost history between the five first order minimisation discussed: Gradient Descent (GD), Momentum (Mo), Nag (Na), Adagrad (Ag) and Adadelta (Ad). These were given the same initial θ to start from, same number of iterations (7500), same learning rate (105) and the same set of layers to fuse (the layers in SNF for the breast cancer dataset, see Section 5.2.2.1)

Figure 4.2 shows a comparison of the cost histories from the five different methods discussed. They were all given the same problem: the same input layers which were the three used by SNF for the breast cancer data set (see Section 5.2.2.1); the same learning rate, number of iterations (105 and 7500) and the same initial starting point. The first thing to note is that apart from Adadelta all of the more sophisticated methods converged faster than normal gradient descent. In fact looking at the cost of final GD iteration, the momentum, NAG and adagrad method become less than this by iteration $963, 923, 675$ respectively. In terms of computational time taken momentum and adagrad took approximately the same time as GD, $6m30s$, whereas NAG took almost twice as long $11m30s$. The precise run times are given in Table 4.1. As explained above this is due to having to calculate and extra gradient at each step making NAG undesirable to use. The results were the same when this analysis was run on the other data sets in SNF. Proceeding forward, all that remains would be to choose which method to use out of Momentum and Adagrad. In terms of distance they are less than 4.5×10^{-6} apart, and as for validation quantities, discussed later in Section 5.1, they give the same values for both solutions, meaning there is little difference between the two solutions and the two methods. Hence we chose to proceed using momentum.

Method	Time
Gradient descent	$6m37s$
Momentum	$6m18s$
NAG	$11m18s$
AdaGrad	$6m22s$
AdaDelta	$6m9s$

TABLE 4.1: The run times for the various minimisation methods shown in Figure 4.2.

4.5 ENF Algorithm

In summary, ENF takes m, $n \times n$ similarity networks as inputs and aims to find the one network that is most similar, in terms of structure, to all of the given inputs, using entropy and the $JS_{\mathcal{N}}$ as proxy for structural equivalence between networks. This process itself is done in a linear regression fashion [77,78] using QJSD, as it is applied in Section 1.3 (from [13]), as a cost function.

The process involves looking at the distance between the given input networks and a initial network, then altering the structure of this inital network to minimise the distances between them. This can intuitively be thought of as finding the network that is closest to *all* of the networks *at the same time*, with θ_{ij}, representing how similar patient i and patient j are across all the layers. A summary of the algorithm can be found in Algorithm 1.

The input of the algorithm is a set of similarity matrices and the number of iterations for the algorithm to run for T. The remaining optional inputs control the momentum gradient descent (learning rate α and momentum decay γ) as well as the initialisation for θ (a random symmetric matrix, a layer, or the average of the input layers).

The algorithm first computes the average entropy of the input layers, as it remains constant for the rest of the process. Next the iterative process begins. In each iteration the algorithm calculates the cost of the current θ and then calculates the derivatives for each entry of the current θ. After this every entry of the momentum term is updated, it is reduced by multiplying by γ and then adding the current derivative multiplied by the learning rate, α. This updated momentum term is then subtracted from θ_{ij} and if any of the entries become less than 0 that entry is set equal to 0, since we only allow non-negative entries. Once the given number of iterations has been reached the process terminates.

Algorithm 1: Entropy Network Fusion

function ENF $(S^{(1)}, \ldots, S^{(l)}, T, \alpha, \gamma, I)$;
Input : l symmetric matrices of size $n \times n$ with entries between 0 and 1.
Number of iteration T.
Learning rate $\alpha \in \mathbb{R}^+$. (Default $\alpha = n$)
Gradient decay $\gamma \in (0, 1)$. (Default $\gamma = 0.9$)
Initialisation I (Default - $I = R$ random, $I = A$ average, $I = k$ layer k)
Output: $n \times n$ symmetric matrix $\theta = [\theta_{ij}]$
if $I \in \{1, \ldots, n\}$ **then**
  $\theta \leftarrow S^{(I)}$
else
  if $I = A$ **then**
    $\theta \leftarrow \frac{\sum_{m=1}^{l} S^{(m)}}{l}$
  else
    $\theta \leftarrow$ random symmetric matrix with entries between 0 and 1
  end
end
$y \leftarrow \frac{1}{l} \sum_{m=1}^{l} H_{\mathcal{N}}(S^{(m)})$
Initialise $\nu = (\nu_{ij})$ to the zero $n \times n$ matrix
for $k = 1, \ldots, T$ **do**
  Calculate the cost $C(\{S^{(m)}\}, \theta)$ as per Eq. 4.2
  Calculate the gradient $\frac{\partial C(\{S^{(m)}\}, \theta)}{\partial \theta_{ij}}$ for each entry as per Eq. 4.20
  for $i = 1, \ldots, n$ **do**
    for $j = 1, \ldots, n$ **do**
      $\nu_{ij} \leftarrow (\gamma \nu_{ij} + \alpha \frac{\partial C(\{S^{(l)}\}, \theta)}{\partial \theta_{ij}})$
      $\theta_{ij} \leftarrow (\theta_{ij} - \nu_{ij})$.
      if $\theta_{ij} < 0$ **then**
        $\theta_{ij} \leftarrow 0$
      end
    end
  end
end

4.6 Convexity

A highly desirable property in any optimisation problem is *convexity*. Mathematically, a real-valued function $f : D \to R$ where $D \subset R^n$, is convex if it satisfies two conditions. The first is that its domain is a convex set. (A set is called convex if the line between any two points of set is also contained in the set.) Second, it satisfies the following:

$$f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha)f(x_2)$$

for all $\alpha \in [0,1]$ and x_1, x_2 in its domain [80]. This means that the line segment connecting the value of the function between any two points is always greater than or equal to the the value of any point in-between. The most important consequence of convexity, for our minimisation purposes, is that the function has a global minimum, and any local minimum found is also a global minimum [80]. If a function is not convex there could be several local minima that are not global minima, and optimisation methods, such as gradient descent, may find different minima depending on the initial starting conditions. Now we show that our cost function for ENF (Eq 4.2) is indeed convex, meaning that there is at most one global minimum and, if it exists, any optimisation method will converge to it

Theorem 4.1. *The cost function of ENF, is convex.*

Proof. Our cost function, which we are going to show is convex, is Eq 4.2:

$$\begin{aligned} C(\theta) &= \frac{1}{l}\sum_{m=1}^{l}\left[H_{\mathcal{N}}\left(tS^{(m)} + s^{(m)}\theta\right)\right] - \frac{1}{2l}\sum_{m=1}^{l}[H_{\mathcal{N}}(S^{(m)})] - \frac{1}{2}H_{\mathcal{N}}(\theta) \\ &= \frac{1}{l}\sum_{m=1}^{l} JS_{\mathcal{N}}(\theta, S^{(m)}) \end{aligned}$$

We write this in terms of θ as the input layers remain constant and we start by showing that $JS_{\mathcal{Q}} = JS_{\mathcal{Q}}(\theta)$ is convex with respect to θ. Recall from Section 3.1, (Eq 3.5), that $JS_{\mathcal{Q}}$ can be written as:

$$JS_{\mathcal{Q}}(\rho,\sigma) = \frac{1}{2}\left[KL_{\mathcal{Q}}\left(\rho\middle|\frac{\rho+\sigma}{2}\right) + KL_{\mathcal{Q}}\left(\sigma\middle|\frac{\rho+\sigma}{2}\right)\right]$$

From [63, 64] we know that $KL_{\mathcal{Q}}(-|-)$ is *jointly convex*. That is let $\rho^{(i)}$ and $\sigma^{(i)}$ $(1 \leq i \leq n)$ denote a set of density matrices and let α_i be a set of non-negative real numbers such that $\sum_i \alpha_i = 1$. Then:

$$KL_{\mathcal{Q}}(\sum_i \alpha_i \rho^{(i)} | \sum_i \alpha_i \sigma^{(i)}) \leq \sum_i \alpha_i KL_{\mathcal{Q}}(\rho^{(i)}|\sigma^{(i)}). \tag{4.24}$$

First, from the definition of $JS_{\mathcal{Q}}$ (Eq 3.5), we see that it contains the term $KL_{\mathcal{Q}}(\rho|\frac{\rho+\sigma}{2})$, so we shall show that this term is convex. Consider a set of density matrices $\rho^{(i)}$ and another set $\sigma^{(i)}$ where $\sigma^{(i)} = \sigma$ for all i. Then:

$$\begin{aligned}
KL_{\mathcal{Q}}(\sum_i \alpha_i\rho^{(i)}|\frac{\sum_i \alpha_i\rho^{(i)}+\sigma}{2}) &= KL_{\mathcal{Q}}(\sum_i \alpha_i\rho^{(i)}|\frac{\sum_i \alpha_i\rho^{(i)}+\sum_i \alpha_i\sigma^{(i)}}{2}) \\
&= KL_{\mathcal{Q}}(\sum_i \alpha_i\rho^{(i)}|\sum_i \alpha_i\frac{\rho^{(i)}+\sigma^{(i)}}{2}) \\
&\leq \sum_i \alpha_i KL_{\mathcal{Q}}(\rho^{(i)}|\frac{\rho^{(i)}+\sigma^{(i)}}{2}) \\
&= \sum_i \alpha_i KL_{\mathcal{Q}}(\rho^{(i)}|\frac{\rho^{(i)}+\sigma}{2})
\end{aligned}$$

As previously discussed, the relative entropy is not symmetric and the case with the terms swapped must also be considered. Given the same conditions:

$$\begin{aligned}
KL_{\mathcal{Q}}(\sigma|\frac{\sigma+\sum_i \alpha_i\rho^{(i)}}{2}) &= KL_{\mathcal{Q}}(\sum_i \alpha_i\sigma^{(i)}|\frac{\sum_i \alpha_i\sigma^{(i)}+\sum_i \alpha_i\rho^{(i)}}{2}) \\
&= KL_{\mathcal{Q}}(\sum_i \alpha_i\sigma^{(i)}|\sum_i \alpha_i\frac{\sigma^{(i)}+\rho^{(i)}}{2}) \\
&\leq \sum_i \alpha_i KL_{\mathcal{Q}}(\sigma^{(i)}|\frac{\sigma^{(i)}+\rho^{(i)}}{2}) \\
&\leq \sum_i \alpha_i KL_{\mathcal{Q}}(\sigma|\frac{\sigma+\rho^{(i)}}{2})
\end{aligned}$$

Now combining these two we show that $JS_{\mathcal{Q}}$ is convex, with respect to ρ, using the same conditions.

$$\begin{aligned}
JS_{\mathcal{Q}}(\sum_i \alpha_i\rho^{(i)},\sigma) &= \frac{1}{2}[KL_{\mathcal{Q}}(\sigma|\frac{\sigma+\sum_i \alpha_i\rho^{(i)}}{2}) + KL_{\mathcal{Q}}(\sum_i \alpha_i\rho^{(i)}|\frac{\sum_i \alpha_i\rho^{(i)}+\sigma}{2})] \\
&\leq \frac{1}{2}[\sum_i \alpha_i KL_{\mathcal{Q}}(\sigma|\frac{\sigma+\rho^{(i)}}{2}) + \sum_i \alpha_i KL_{\mathcal{Q}}(\rho^{(i)}|\frac{\rho^{(i)}+\sigma}{2})] \\
&= \sum_i \alpha_i \frac{1}{2}[KL_{\mathcal{Q}}(\sigma|\frac{\sigma+\rho^{(i)}}{2}) + KL_{\mathcal{Q}}(\rho^{(i)}|\frac{\rho^{(i)}+\sigma}{2})] \\
&= \sum_i \alpha_i JS_{\mathcal{Q}}(\rho^{(i)},\sigma)
\end{aligned}$$

Hence we see that $JS_{\mathcal{Q}}(\theta)$, and $JS_{\mathcal{N}}(\theta)$, is convex with respect to θ. Using this we can now construct our cost function.

$$\begin{aligned}
C(\alpha_1\theta_1 + \alpha_2\theta_2) &= \frac{1}{l}\sum_{m=1}^{l} JS_{\mathcal{Q}}(\alpha_1\theta_1 + \alpha_2\theta_2, S^{(m)}) \\
&= \frac{1}{l}JS_{\mathcal{Q}}(\alpha_1\theta_1 + \alpha_2\theta_2, S^{(1)}) + \cdots + \frac{1}{l}JS_{\mathcal{Q}}(\alpha_1\theta_1 + \alpha_2\theta_2, S^{(l)}) \\
&\leq \frac{\alpha_1}{l}JS_{\mathcal{Q}}(\theta_1, S^{(1)}) + \frac{\alpha_2}{l}JS_{\mathcal{Q}}(\theta_2, S^{(1)}) + \cdots + \frac{\alpha_1}{l}JS_{\mathcal{Q}}(\theta_1, S^{(1)}) \\
&\qquad + \frac{\alpha_2}{l}JS_{\mathcal{Q}}(\theta_2, S^{(1)}) \\
&= \alpha_1\frac{1}{l}\sum_{m=1}^{l} JS_{\mathcal{Q}}(\theta_1, S^{(m)}) + \alpha_2\frac{1}{l}\sum_{m=1}^{l} JS_{\mathcal{Q}}(\theta_2, S^{(m)}) \\
&= \alpha_1 C(\theta_1) + \alpha_2 C(\theta_2)
\end{aligned}$$

Which is precisely the condition for convexity. Therefore we have shown that our cost function for ENF, and wENF, is convex too. □

4.7 Computational Time Analysis

To evaluate how our algorithm scales with respect to the input data, we performed a numerical analysis to evaluate its computational complexity. In our experiment, we generated the graphs to be integrated randomly with an edge probability of 0.7, and random initial θ. We repeated the experiment 100 times for each combination of vertices and layers, and the collection of time values averaged, the results of which can be seen in Figure 4.3. From this it can be seen that the time complexity of the algorithm seems linear on the number of layers, which is to be expected as the same calculations are being repeated for the extra layers. As for the increase in vertices, Figure 4.3 shows an increase that seems between a quadratic and cubic form, which is approximately the complexity of computing eigenvalues and eigenvectors of symmetric matrices, the most time-consuming part of our algorithm.

4.8 Convergence

In this section we report some validation tests for the ENF algorithm. In particular, we test the effects of different learning rates on the algorithm convergence; that the output from ENF produces a non-trivial answer containing useful information; the variance arising from different initialisations of θ; the speed of convergence when starting from the average graph and that ENF converges to a non-trivial answer.

In the first numerical test we designed, we generated 100 graphs on a given number of vertices and then ENF performed on them individually. Their cost histories were then averaged and plotted along with error bars equal to one standard deviation. We repeated this for a selection of learning rates and then further on a different number of vertices. The results are shown in Figure 4.4 and as can be seen the cost does converge/decrease in every example. However it is curious to note that the convergence rapidly accelerates as the learning rate increases, suggesting that an optimal learning rate could be larger than 1. Typically in ML the learning rate is kept small as a learning rate that is too large can cause the cost to diverge and increase rather than converge to a minimum [77, 78]. However with ENF we have found that a larger learning rate is required to have the method converge in an acceptable time. Later in Section 5.2.2 when applying it to real data we found an acceptable value for α was the number of node in the networks.

Another technique we will use to help visualise the results of ENF is Multidimensional Scaling (MDS) [84] to embed the networks into $\mathbb{R}^2$. Given a set of distances MDS finds the best set of points in a lower dimensional space ($\mathbb{R}^2$ in our case) whose distances are as close as possible to the distances provided. When plotting using these lower dimensional coordinates we refer to the axes as the MDS axes. The distances used in MDS, in this case, were the distances between the networks, the square-root of $JS_{\mathcal{N}}$. An example of this embedding can be found in Figure 4.5. In this example 8 graphs on 20 were fused together: A bipartite graph with a $10 - 10$ split; the complete graph; an Erdos-Renyi block graph $p = 0.4, q = 0.05$; the path graph; A random graph with edge connection $p = 0.34$; the ring graph; the star graph and a random tree graph. The average of these graphs was also embedded and coloured red to highlight its position. Alongside these, the evolution of the solution from ENF was also embedded and coloured with respect to the cost at that point. As can be see from the figure the solution passes the average graph indicating ENF producing a non-trivial answer, different from the average of the networks. Knowing that ENF converges to a solution, next we tested if the output being produced was a sensible answer. To test this, we gave ENF a single network, initialised randomly, as a input and then evaluated the results. In theory, as there is only one input, the result should be the same network. This test was run multiple times with different graphs obtaining the same results, therefore only one set of results are shown and discussed. These are shown in Figure 4.6, ENF was implemented with 400 iterations and $\alpha = 20$. As the input network was unweighted, shown in Figue 4.6 A, the output network was rescaled such that its maximum weight equaled 1. Figure 4.6 B and C show the cost history from the process and the MDS embedding of the iterations, which is explained shortly. From Figure 4.6 D we can see that the output network is virtually the same except with some extra, extremely weak, edges. In total there are 13 extra edges whose mean weight is 1.9×10^{-4} and maximum weight of 6.7×10^{-4}. When these edges are thresholded out, the structure becomes identical to the input graph. The other edges have a mean weight of 0.9997 and a standard deviation of 2.4×10^{-4}, making them extremely consistent and virtually equal.

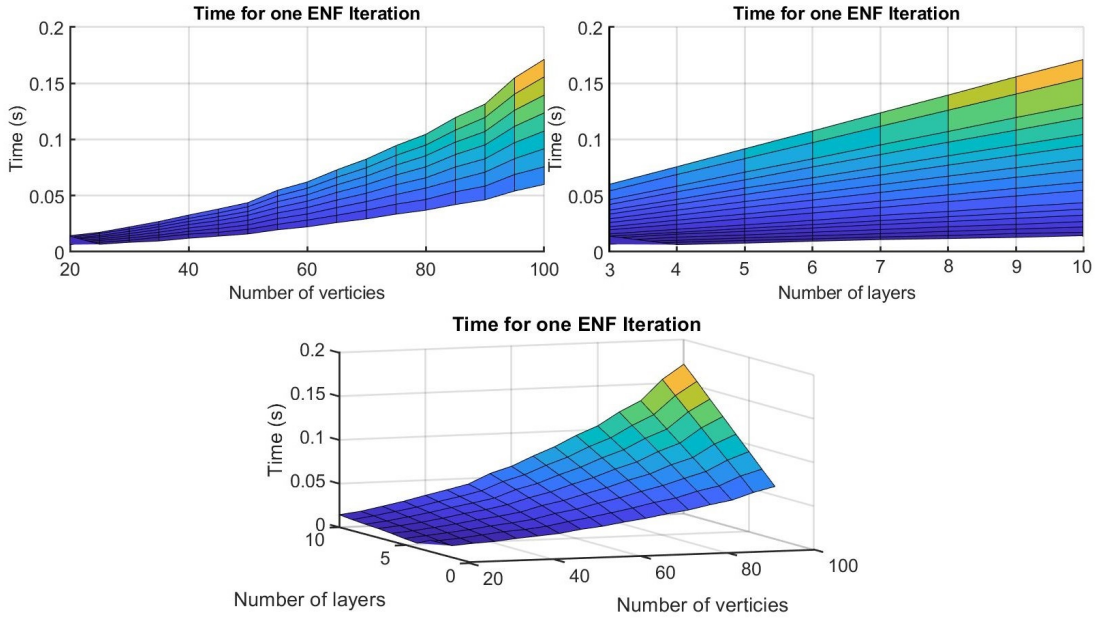

FIGURE 4.3: Shows the results from the time analysis of ENF. For each combination of vertices and layers, we timed 100 instances of one iteration of ENF and then plotted the average time, colouring it proportionally to its value. The computer system used to perform this test was the Iridis 4 computer cluster [83] and the test was computed in parallel to reduce the overall time of the test. Specifically each node possessed 16 cores and allocated 4 Gb of RAM.

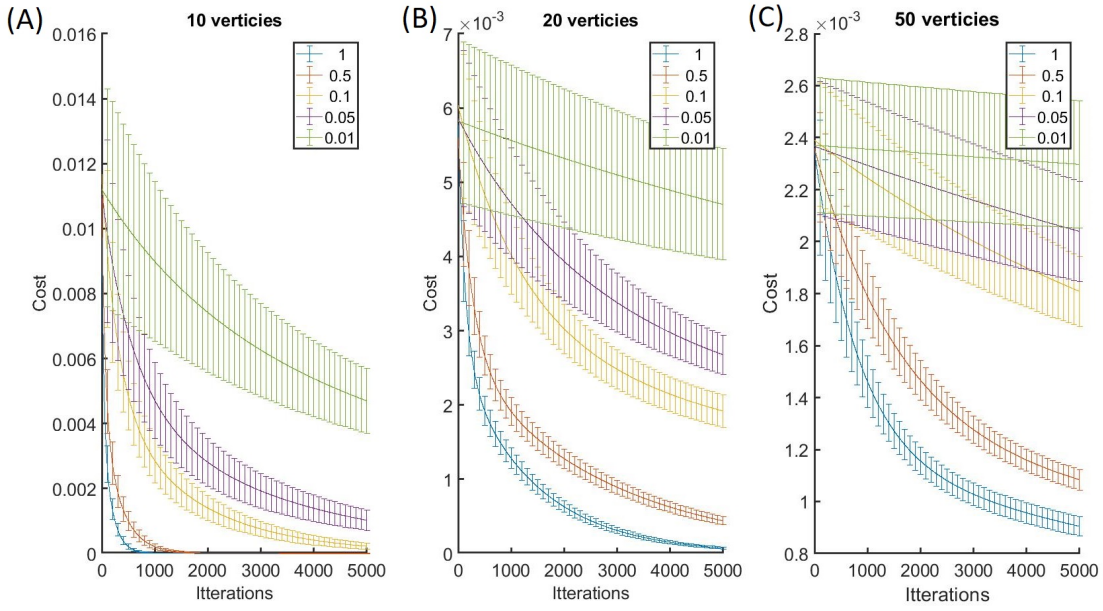

FIGURE 4.4: The results from the convergence check of the ENF algorithm. For each setting of vertices and learning rate ENF was implemented on 100 different graphs, one at a time. The resulting cost histories were then averaged and plotted with one standard deviation. The legends show the value of the learning rate used in each setting. (A) Graphs with 10 vertices. (B) Graphs with 20 vertices. (C) Graphs with 50 vertices.

Next, given that we typically inititalise ENF at a random start point, we tested the consistency in the output of ENF. To do this we implemented ENF 100 times with the same single target network. Each output was rescaled, such that the maximum value in each adjacency matrix became equal to one, an acceptable adjustment because as shown ENF is scale independent, and allows for the outputs to be more easily compared. To measure the consistency we looked at the variance of each entry across all of the outputs. If all the entries have a low variance then we can conclude that ENF is consistent. A full boxplot of the entry-wise variances, as percentages of the mean entry-wise value, is shown in Figure 4.7. As we can see the results showed variance in the range of $[0.09\%, 1.3\%]$, including the ones classed as outliers. This means that, regardless of the initial point (random), it converges to the same target graph with small variance across repetitions, showing that ENF is incredibly consistent with respect to the initialisation.

As we have been initialising ENF at a random θ we ran a subsequent test to look at the effect of initialising θ at the average instead. In this, 6 graphs with 25 vertices were randomly generated and then we implemented ENF twice on these same networks, once with θ being initialised randomly, (and symmetrically) and the other with θ initialised at the average of the given graphs, both using a learning rate of $\alpha = 2.5$. This was repeated 100 times, averaging all of the cost history and plotting it with error bars equal to one standard deviation. The results can be seen in Figure 4.8 which clearly shows that the implementation with the average initialisation converges faster. However testing found that if any one of the networks have values on a significantly different scale compared to the others then this can in fact slow down the convergence, as the average would be biased towards one of the input networks.

We also validated ENF is some biomedical data sets, this is discussed in Chapter 5.

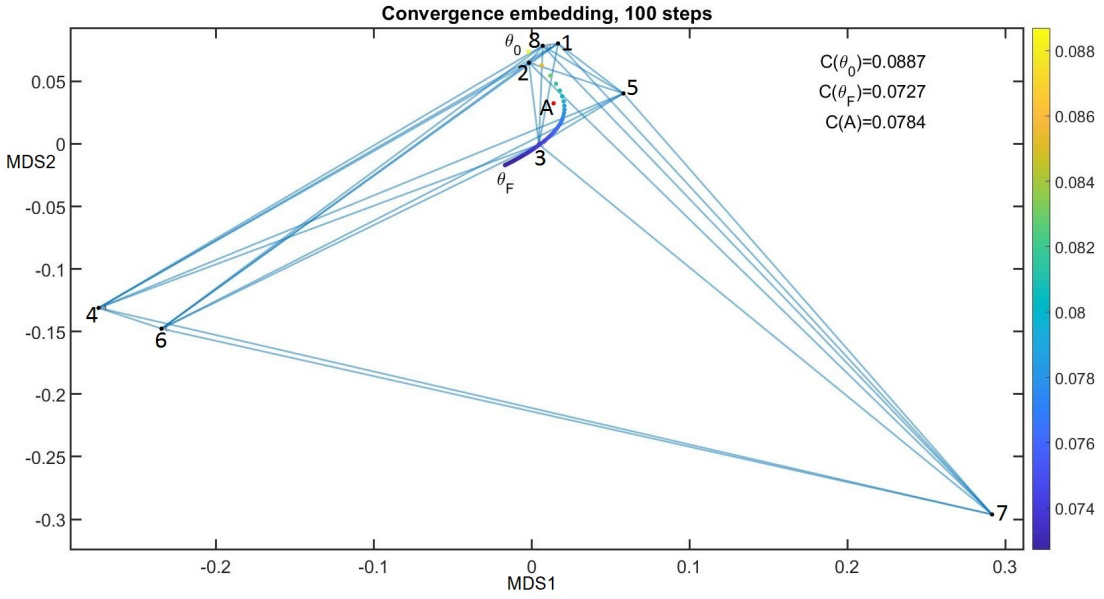

FIGURE 4.5: Common graphs embedded with MDS using the square-root of $JS_{\mathcal{N}}$. The graphs used: (1) Bipartite graph; (2) Complete graph; (3) Erdos-Renyi block graph; (4) Path graph; (5) Random graph; (6) Ring graph; (7) Star graph and (8) Tree graph. The long edges between the input graph are to illustrate the distance between them. The red point is the average graph with its cost shown in the top right. The trail of colour points is the evolution of the ENF output, every 100 iterations, coloured by its cost at that point. The initial cost and the cost at the final iteration is shown in the top right. This helps to illustrate that our method is finding a non trivial answer.

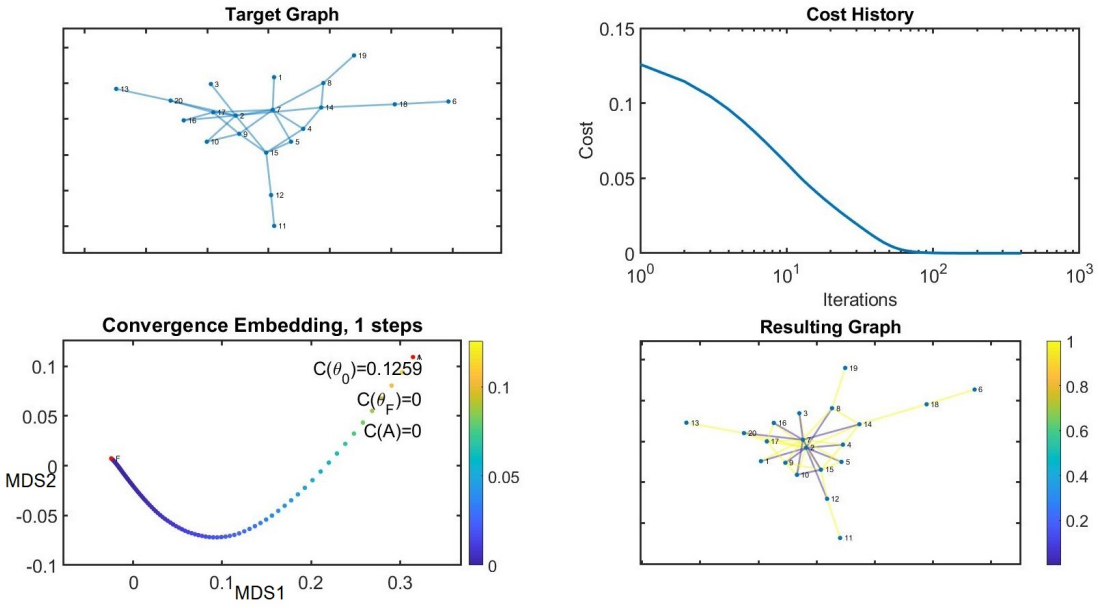

FIGURE 4.6: The results from our test that ENF recovers the input graph when there is only one layer. (A) Plot of the small simple graph given to ENF as (single layer) input. (B) The cost history from ENF. (C) The MDS embedding of the evolution of the solution for every step, (see the main text for more details). (D) Plot of the output network from ENF with its edges coloured proportional to its edge weight. As we see the original input graph is recovered with a couple of extra, very weak edges.

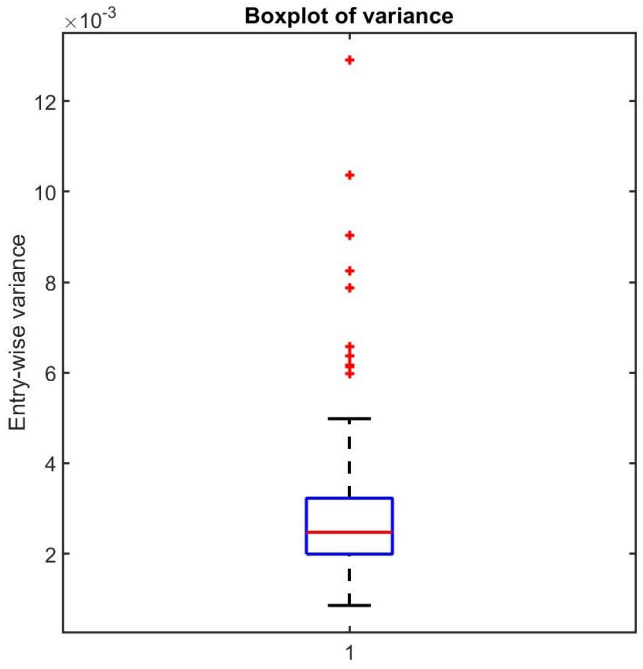

FIGURE 4.7: Results from our initialisation consistency test. ENF was run 100 times on the same graph (20 vertices), each time from a different random initialisation. After rescaling such that the maximum value was equal to 1, the entry-wise variance was calculated across all 100 instances. These variances have been plotted as a percentage of the mean entry-wise value.

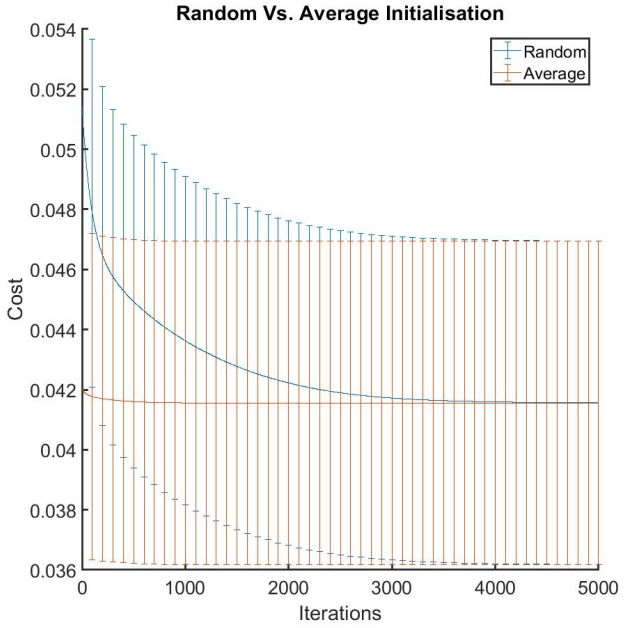

FIGURE 4.8: Shows the difference in speed of convergence when the average of the input layers is used for the initialisation of θ instead of a random one. For each setting 100 implementations of ENF were done and the average cost history, with one standard deviation, have been plotted.

Chapter 5

Validation and Results on cancer datasets

After defining our novel network integration method, and some initial validation tests, we will now apply ENF to real biomedical data to evaluate its performance. To allow a side-by-side comparison with SNF, we will apply our algorithm to the same cancer data sets in [14] when presenting SNF. This Chapter is divided into two Sections. The first section explains the validation tools we shall use: abstract cluster comparison metrics, and survival analysis (for the survival metadata in the cancer sets). The second section contains the results and analysis from the application of ENF to the SNF cancer datasets and the comparison with the SNF results. We will also compare our results with simpler Data Integration methods (concatenation, averaging and random clustering).

5.1 Validation Tools

The Section covers the tools that will be used to evaluate and validate the results of our algorithm. We will discuss survival analysis, as well as various metrics used to compare clustering solutions.

5.1.1 Survival Analysis

Survival analysis consists of techniques to analyse data where the time until an event occurs is of interest [56, 85]. The event in question can be patient time until recovery, readmission or, as in this case, death. The time variable, t, is usually referred to as *survival time*. These techniques also make use of what is known as *censored data*. Censored data is where the exact survival time is unknown but, at the time recorded, the subject was alive. Reasons why the exact survival time may be unknown can be the

study ended or before exiting the study [56, 85]. This survival data can be expressed as a pair (t_i, o_i), where i is the subject index, t_i is the time until the outcome o_i, which is equal to one if death occurred and zero if the data is censored.

Given this survival data a *survival function* or *survival curve*, $S(t)$ can be constructed. This survival function gives the probability that a person survives longer than time t [56,85]. Theoretically, this is a smooth monotonically decreasing function where $S(0) = 1$ and $S(\infty) = 0$, but in practice this is not necessarily the case. This function has to be estimated from the data observed and as a result a step function, $\hat{S}(t)$ which will be defined shortly, is obtained which is not a smooth function. Also, studies run for a fixed length of time T and the event may not occur in that time frame for some individuals, giving rise to censored data, hence $\hat{S}(T)$ may not equal 0 [56, 85].

A popular method for estimating the survival function is the *Kaplan-Meier* estimator. In summary, the observed data is used to estimate the survival probability at each time t. This estimated probability, $e(t)$, is defined as:

$$e(t) = \frac{n_t - d_t}{n_t}, \tag{5.1}$$

where n_t is the number of people who are alive at time t and d_t is the number of deaths that occurred at time t [56, 85]. If a censored observation and a death occur at the same time then it is considered that the censored observation occurs 'after' the death (the censored subject is included in n_t). These are then multiplied together to obtain an estimate of the overall survival function [56, 85]:

$$\hat{S}(t) = \prod_{k=1}^{t} e(k). \tag{5.2}$$

These can then be plotted to obtain curves that will be seen later in this chapter, typically when being plotted a cross will be marked on the curve at each censored data point.

The survival curves can be analysed visually to extract information including estimated percentile information. To do this graphically, given a percentile q, a horizontal line is drawn at hight q on the y-axis (the survival probability). At the point where this horizontal line first intersects the survival curve a vertical line is drawn and the corresponding point on the x-axis is the time for that percentile [56]. As previously discussed our survival curve, $\hat{S}(t)$, is an estimated function which may not reach zero by the end of the study $\hat{S}(T)$. Therefore it may not be possible to calculate some percentiles as the horizontal line will not intersect the curve. For example a survival curve that ranges from 1 to 0.45 can estimate the 46th percentile but not the 44th. These percentiles can be formally written [56] as:

$$\hat{t}_q = \min\left\{t : \hat{S}(t) < \frac{q}{100}\right\} \tag{5.3}$$

Once a survival function has been constructed it can then be compared to other survival functions. For example the patients can be grouped by the treatment they receive and then a survival function can be made for each group. Conclusions about the effectiveness of this treatment can then be inferred from the difference in survival functions. These different survival curves can then be tested to see if they are statistically equivalent, a popular choice being the *log-rank test*. The null hypothesis is that all the survival curves are the same curve [56, 85]. The log rank test in essence compares the number of observed deaths to the number of expected deaths [56, 85], as follows. Let us assume the case when there are two groups. Using the same notation as above let n_{ij} denote the number of people alive at time i in group j and let d_{ij} be the number of those that died at time i in group j. Then the expected number of deaths at time i for group j is equal to

$$e_{ij} = \frac{n_{ij}}{n_{i1} + n_{i2}}(d_{i1} + d_{i2}). \tag{5.4}$$

The total difference between the observed and expected for group j is given by

$$O_j - E_j = \sum_i (d_{ij} - e_{ij}) \tag{5.5}$$

This is then used to form the statistic for the log-rank test, which is given by:

$$LR = \frac{(O_j - E_j)^2}{Var(O_j - E_j)} \tag{5.6}$$

where

$$Var(O_j - E_j) = \sum_i \frac{n_{i1}n_{i2}(d_{i1} + d_{i2})(n_{i1} + n_{i2} - d_{i1} - d_{i2})}{(n_{i1} + n_{i2})^2(n_{i1} + n_{i2} - 1)}$$

This value, LR, is then tested under the χ^2 distribution with 1 degree of freedom to give a p-value. If the p-value is less than the given threshold then the null hypothesis (the survival curves are the same) is rejected. For K groups it becomes more involved, using covariances between the different $O_j - E_j$ and the LR is considered under a χ^2 distribution with $K - 1$ degrees of freedom. Exact details can be found in [56, 85]. In the log rank test all parts of the curve are weighted equally but there are other tests which apply weights to different parts of the curve. For example, the *Peto* test applies a weight equal to the number of those at risk at the given time. This results in the failures at the start receiving more weight than those at the end [56, 85]. Proceeding forward the log rank test will be used so all parts of the curvival curve are considered equally.

$U^T V$	V_1	V_2	...	V_C	Σ
U_1	n_{11}	n_{12}	...	n_{1C}	a_1
U_2	n_{21}	n_{22}	...	n_{2C}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
U_R	n_{R1}	n_{R2}	...	n_{RC}	a_R
Σ	b_1	b_2	...	b_C	N

TABLE 5.1: The number of elements that are in cluster i of U and cluster j of V, $|U_i \cap V_j|$, is given by n_{ij}. The row and column sums are denoted by a_i and b_i respectively.

5.1.2 Clustering Comparison

This Section will cover the quantities that shall be used to evaluate the clustering results of our method and any others we compare against. Where survival analysis can only be done in the presence of survival data, these following quantities can be used in any abstract network clustering situation. In particular we introduce network quality measures: Normalised Variance of Information (NVI), Normalised Information Distance (NID) and Concordance Index (CI). We will use them to compare the output from the various fusion methods, hence some are needed tools to compare the clusterings obtained.

First, we discuss NVI and NID following [86]. These quantities are metrics on clusterings on the same graph. Having a metric enables us to directly compare clustering outputs. These metrics use Information Theory and Entropy to evaluate how much information two clusterings share: the more information they share, the closer they are.

Consider two clusterings, one that has R clusters and the other having C. Let us use U and V as their clustering indicator matrices which are of order $(n \times R)$ and $(n \times C)$ respectively. The entries of these matrices are binary, so either 1 or 0, where the (i, j)-entry indicates if node i is a member of cluster j and each node is only a member of one cluster. The first step to calculating NVI and NID is to look at their table of intersections, that is, how many pairs of clusters overlap. This can be read off the product $U^T V$ a matrix of order $(R \times C)$, where the (i, j) entry is the number of elements that are in both cluster i of U and cluster j of V, $|U_i \cap V_j|$. This gives a table/matrix like the one shown in Table 5.1. From this table the following functions can be defined: $H(U)$ (Eq 5.7) is the entropy of the clustering U, to avoid confusion this shall be referred to as the *cluster entropy*; $H(U, V)$ (Eq 5.8) is the *joint entropy* and $I(U, V)$ (Eq 5.9) is

the *mutual information* (MI).

$$H(U) = -\sum_{i=1}^{R} \frac{a_i}{N} \log_2 \left(\frac{a_i}{N}\right), \tag{5.7}$$

$$H(U,V) = -\sum_{i=1}^{R}\sum_{j=1}^{C} \frac{n_{ij}}{N} \log_2 \left(\frac{n_{ij}}{N}\right), \tag{5.8}$$

$$I(U,V) = \sum_{i=1}^{R}\sum_{j=1}^{C} \frac{n_{ij}}{N} \log_2 \left(\frac{Nn_{ij}}{a_i b_j}\right). \tag{5.9}$$

In [86] the authors state that MI, as it is non-negative, can be used as a basic similarity measure however they remark that a desirable feature of any cluster comparison is normalisation. Given that MI is bounded by:

$$\begin{aligned} I(U,V) \leq \min\{H(U), H(V)\} \leq \sqrt{H(U)H(V)} \leq \frac{1}{2}(H(U) + H(V)) \leq \dots \\ \leq \max\{H(U), H(V)\} \leq H(U,V) \end{aligned} \tag{5.10}$$

multiple normalised measures can be constructed [86]. When these bounded quantities are turned into normalised distance measures, only two turn out to be metrics. The *Normalised Variation of Information* (NVI), Eq. 5.11 and the *Normalised Information Distance* (NID) Eq. 5.12.

$$NVI(U,V) = 1 - \frac{I(U,V)}{H(U,V)} \tag{5.11}$$

$$NID(U,V) = 1 - \frac{I(U,V)}{\max\{H(U), H(V)\}} \tag{5.12}$$

Therefore, the closer to zero these quantities are, the more similar they are. For example, consider two clusterings that are identical but one has had its labels permuted. These would have a distance of zero because they have the same information/structure. The table of intersection in this case would only have one entry in each row/column that is non-zero and we can re arrange it to look like the table below

$U^T V$	V_1	V_2	$\dots$	V_C	Σ
U_1	a_1	0	$\dots$	0	a_1
U_2	0	a_2	$\dots$	$\vdots$	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	0	$\vdots$
U_C	0	$\dots$	0	a_C	a_C
Σ	a_1	a_2	$\dots$	a_C	N

At this point, the terms $H(U), H(V)$ and $H(U,V)$ are all equal. Therefore we just need to show these are equal to $I(U,V)$ and in this case, the non-zero terms in MI become

$$\frac{n_{ij}}{N}\log_2\left(\frac{Nn_{ij}}{a_i b_j}\right) = \frac{a_i}{N}\log_2\left(\frac{Na_i}{a_i a_i}\right) = \frac{a_i}{N}\log_2\left(\frac{N}{a_i}\right) = -\frac{a_i}{N}\log_2\left(\frac{a_i}{N}\right)$$

making the summation equal to that of $H(U), H(V)$ and $H(U,V)$. Hence the distance between these two clusterings will be equal to 0. Whilst on the opposite end, a clustering with one single cluster and another where every point is a cluster will have a maximal distance of one. To see this, consider the table of intersections in this scenario, which would look like the one below. Note that as we have only one row, each n_{1j} is in fact

$U^T V$	V_1	V_2	$\ldots$	V_C	Σ
U_1	b_1	b_2	$\ldots$	b_C	N
Σ	b_1	b_2	$\ldots$	b_C	N

equal to its column sum b_j. To show in this situation the NID and NVI are equal to one, we only have to observe that the MI, $I(U,V)$, is zero. The fraction inside the logarithm of MI, $\frac{Nn_{ij}}{a_i b_j}$, in this situation it becomes $\frac{Nb_j}{Nb_j} = 1$ and therefore the logarithm is zero for all the terms. As QJSD is being used to find a network that is closest to the structure of all of the given input networks, the above metrics shall be used to compare the clustering of the ENF output to the clustering of each of the input networks.

In [14], the authors use Concordance Index (CI) to evaluate clusterings and, to keep the comparison fair, this is included in our analysis as well. The CI takes two clusterings, U and V, and measures how much agreement there is. It is bounded between 0 and 1, where the closer to 1 the more agreement between the clusterings. The CI can be defined from some of the quantities defined above as

$$CI(U,V) = \frac{I(U,V)}{\sqrt{H(U)H(V)}}. \tag{5.13}$$

This is simply the MI normalised by one of the bounds shown in Eq. 5.10. However, as shown in [86], this quantity is not a metric, unlike NVI and NID, but since the CI is not used as a metric this is not an issue.

5.2 Results from SNF Data

This Section contains results of various methods being applied to the cancer data sets from [14] and a survival analysis performed. There were five data sets available for analysis: Lung, Colon, GLIO, Breast and Kidney, each of which had data from three sources: mRNA expression, DNA methylation and miRNA expression. Once integrated together, the results were clustered, via spectral clustering [53,54]. This, along with the survival data that was also available for each data set, allows for a survival analysis to be

performed [56], to evaluate the difference in survival profiles. Additionally, we include the results from two naive approaches: average and concatenation (Section 5.2.1).

5.2.1 Naive Methods

Before evaluating SNF and ENF, we apply two naive methods for DI to demonstrate why more sophisticated methods are needed. The first is a intermediate DI method, which is averaging: The output is simply the average of the similarity matrices. The second, an early DI method, is Concatenation: Before any similarity matrix is formed the data is concatenated into one data matrix and then a single similarity matrix, the output, is formed. Concatenating the data removes all the partitioning that has naturally arisen from the different sources and force the similarity measure to consider everything at the same time. With the results from these naive methods they were then clustered, using spectral clustering, into the same number of clusters that were used in SNF.

Upon construction of the average similarity matrix for the breast data, the resulting matrix has diagonal entries equal to one and off diagonal entries that are all very similar in value. Examination of these off diagonal entries show that they have a mean on the scale of 10^{-8} and a variance that is less than 10^{-10}, which could be the result of a normalisation issue. Results like this are also obtained when the other cancer data sets from SNF [14] are used. The cluster profiles from when these matrices are clustered are shown in Table 5.2.

Data	Number of clusters	$\|C_1\|$	$\|C_2\|$	$\|C_3\|$	$\|C_4\|$	$\|C_5\|$
Kidney	3	120	1	1	-	-
Breast	5	101	1	1	1	1
Colon	3	90	1	1	-	-
Lung	4	103	1	1	1	-
Glio	3	212	1	2	-	-

TABLE 5.2: After integrating the desired data set by averaging the similarity matrices the results were then clustered into the corresponding number of clusters used in SNF. This table shows the number of members in each cluster for each data set.

When these are clustered into k clusters, using spectral clustering, it mostly results in $k-1$ clusters containing one patient and the remaining patients in the last clusters. This means that the method does not find any significant structure in the data. This is further backed up by the p-values shown in Table 5.3 as unsurprisingly, none of the survivial curves have a significant p-value. Therefore these results are of no practical use, and simply aggregating the layers as a fusion method is ineffective.

For concatenation it can be seen from Table 5.3 that in all but one case, the concatenation of the glio data, these naive methods do not tend to give significant solutions. This supports the need for more sophisticated methods for combining the data.

Data	No. Clusters	Aggregation p-value	Concatenation p-value
Breast	5	0.981	0.331
Lung	4	0.1	0.639
Kidney	3	0.834	0.912
Colon	3	0.753	0.516
Glio	3	0.208	0.00121

TABLE 5.3: p-values from the log rank test on the SNF data after being integrated by our two naive methods. The results from these data sets were then clustered according to the corresponding number of clusters used in SNF.

We also compare our results to a random clustering. For each dataset, we clustered the patients in to k clusters 1000 times for each $k = 2, \ldots, 5$ and using the survival data p-values calculated for each clustering. For each k, all of the p-values were then averaged and the standard deviation calculated which are shown in Table 5.4. As we can see, within one standard deviation, no results fall below the 0.05 significance threshold. Not only does this show that it is highly unlikely to randomly obtain a significant clustering, but it also gives a measure to compare against. If the p-values obtained from a method are not outside this range then it is no better than clustering randomly.

	2		3		4		5	
Data	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Breast	0.4808	0.2932	0.4857	0.2893	0.4708	0.2934	0.4687	0.2962
Colon	0.5102	0.2825	0.4847	0.2922	0.493	0.2844	0.4762	0.2758
Glio	0.8002	0.2916	0.5062	0.2923	0.4898	0.2927	0.476	0.2841
Kidney	0.4996	0.291	0.4863	0.2923	0.496	0.2864	0.4757	0.289
Lung	0.5072	0.289	0.5004	0.2933	0.4888	0.29	0.4957	0.2952

TABLE 5.4: Average p-values and standard deviations from clustering the patients randomly into 2, 3, 4 or 5 clusters.

5.2.2 ENF Results

In this Section, we apply ENF and SNF to each of the cancer data sets from [14] and make a side by side comparison of the results. To make a fair and effective comparison between SNF and ENF, the similarity networks formed and used in the SNF process were duplicated and then given to ENF as its inputs. For all of the data sets, ENF was implemented using a learning rate equal to the number of patients, and for the number of iterations we used $7,500$ for Breast, Colon, Lung, $10,000$ for Kidney and $15,000$ for Glio. The number of iterations increased for Kidney and Glio as they had more patients than the other three datasets and therefore require more time to converge. All of the computations were done using Matlab 2017b.

The following 5 subsections, one for each data set, follow the same structure. They begin by visualising the input similarity networks, then the information from the fusion process. This consists of the cost history given on a log scale, the embedding of the evolution of the ENF output with the input networks, followed by a side by side comparison of the ENF and SNF output. The distributions of the edge weights are also considered to see which method is less noisy out of the two solutions. After this, we then show tables summarising the results for each of the validation metrics, for both methods. The subsections will then conclude with the survival plots and quartiles for the clusters identified in ENF and the results from the log rank test on both outputs.

In terms of computational speed, SNF performs 20 iterations by default, completing the computation for each cancer dataset in under a minute. On the other hand, with the values that we use for ENF the time it takes to complete is longer: Breast dataset 6m18s; Colon dataset 4m24s; Glio dataset 103m37s; Kidney dataset 11m50s; Lung dataset 6m35s. However, as we are using gradient descent, we can modify the learning rate to decrease the number of iterations and hence the time required for the completion of ENF on a given dataset.

5.2.2.1 Breast Cancer

First, we show a colour plot of the input matrices (Figure 5.1). Note that layer 1 and 2 are on the scale 10^{-5} whilst layer 3 is on the scale of 10^{-3}. Because of this we selected a random initialisation of θ rather than an average initialisation. Otherwise the third layer dominates the initialisation and as a consequence slows down the convergence.

Next the cost history is shown in Figure 5.2 so we can see the convergence of the method. The cost initialised at 0.0453 and after the fixed number of iterations finished at 0.0173, giving a reduction of 62%. For comparison, when we put the SNF solution into the ENF cost function it achieved a cost of 0.0323, almost twice as much as our ENF solution. We define the *delta-change*, δ, as the difference between the cost at the final iteration and the cost 1000 iterations before (this is to gain a sense of how well converged the current solution is). In this case the delta change is $\delta = -4.2 \times 10^{-11}$. Given how small this change is, and how flat the tail end of Figure 5.2 is, this would indicate that the method has (numerically) converged. Upon completion of ENF we embedded the result, its evolution and the input networks into 3 dimensions as discussed in Section 4.8. This is so that the method and its progression can be visualised and given some context in relation to the inputs. This embedding we show in Figure 5.3, the points labeled 1 - 3 correspond to the input layers and the red point labeled 'A' corresponds to the average graph. The average graph is closest to the third layer, which makes sense as it was on a largest scale and dominates the average. However, it is not close to the centre, where ENF finds the minimum cost. In Euclidean space, the point that minimises the average of squared distances for a collection of points is its barycentre, or arithmetic average of all the points. For a triangle this is in its interior near the middle, which is where our intuition suspected the ENF solution would find the minimum solution.

Lastly, in Figure 5.4 we show the output of ENF side by side with SNF so that the difference in the two results can be observed. Note the diagonal of SNF is all ones and had to be removed otherwise the off diagonal structure would not be able to be seen given how small they are in comparison. This was not an issue for ENF but was also done to keep the comparison fair. The same results occured for the other data sets and the same response was taken. Here we see that the structure of the ENF solution is clearer and more well defined than that of the SNF solution. In terms of time taken for ENF to complete, using α equal to the number of patients, 105, and 7500 iterations, ENF took 6 minutes and 18 seconds to complete.

Having the results from both SNF and ENF we now evaluate both results using the various metrics discussed in this thesis. For the evaluations that require a clustering, both solutions were clustered into the same number of clusters used in the SNF paper [14]. As for the method of clustering used, we used spectral clustering as this was the method used in the SNF paper. For this data set, the number of clusters we used was 5. If the number of clusters were to be chosen according the number of clusters that arose

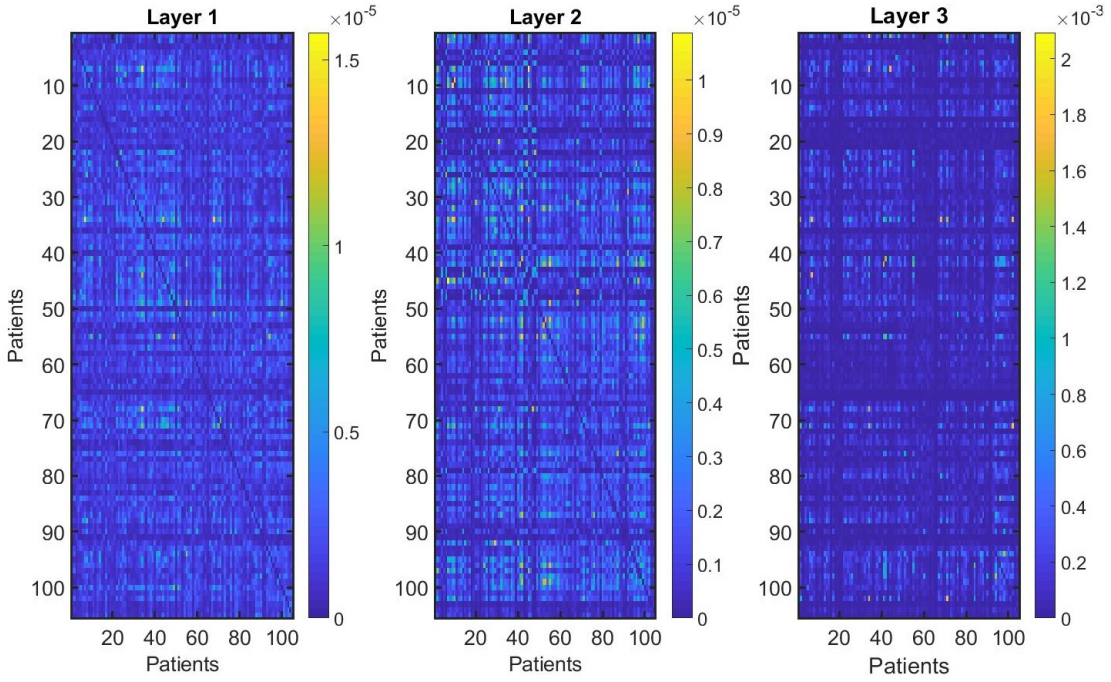

FIGURE 5.1: Heatmaps of the three adjacency matrices that were given as inputs to ENF for the Breast dataset. These initial inputs were created by the SNF method and so were given to ENF to keep our comparison as fair as possible. Layer 1 is mRNA expression, Layer 2 is DNA methylation and Layer 3 is miRNA expression.

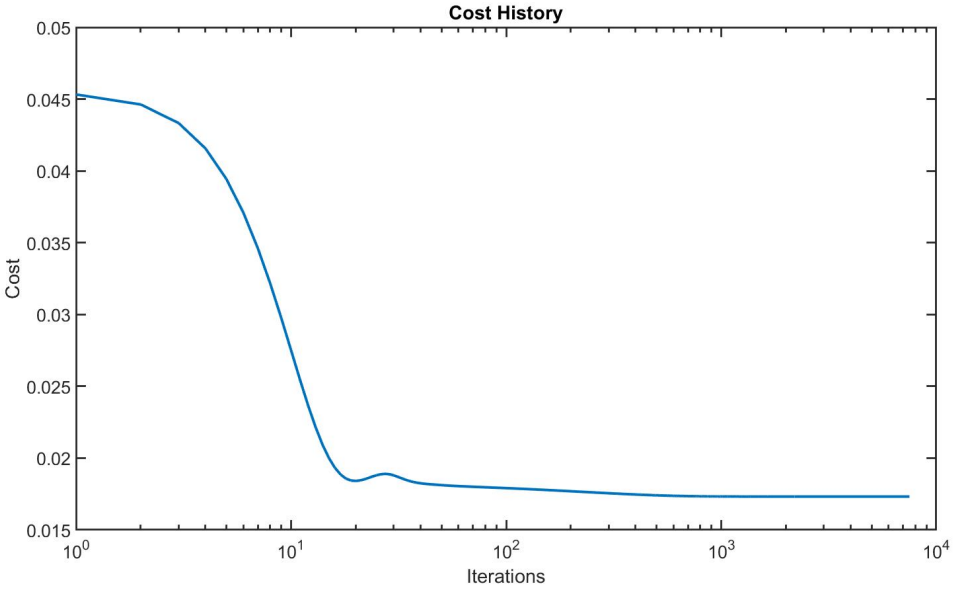

FIGURE 5.2: The value of the ENF cost function at each iteration of the integration process.

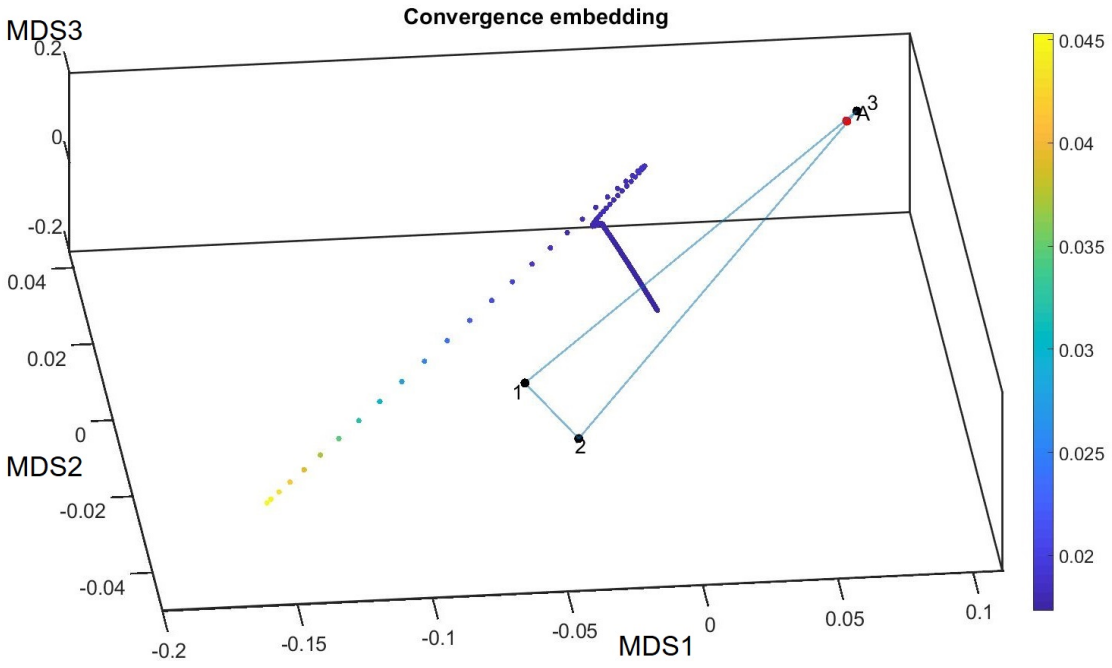

FIGURE 5.3: A visualisation of the networks embedded in $\mathbb{R}^3$ using MDS and the $\sqrt{JS_{\mathcal{N}}}$ as distances. The points labeled 1, 2 and 3 are the input networks shown in Figure 5.1; the point labeled 'A' is the average of the input networks; the remaining points are the complete history of theta coloured with respect to the cost at that time. Note that the ENF converges to the approximate barycenter of the triangle (the point equidistant to 1, 2 and 3), as expected.

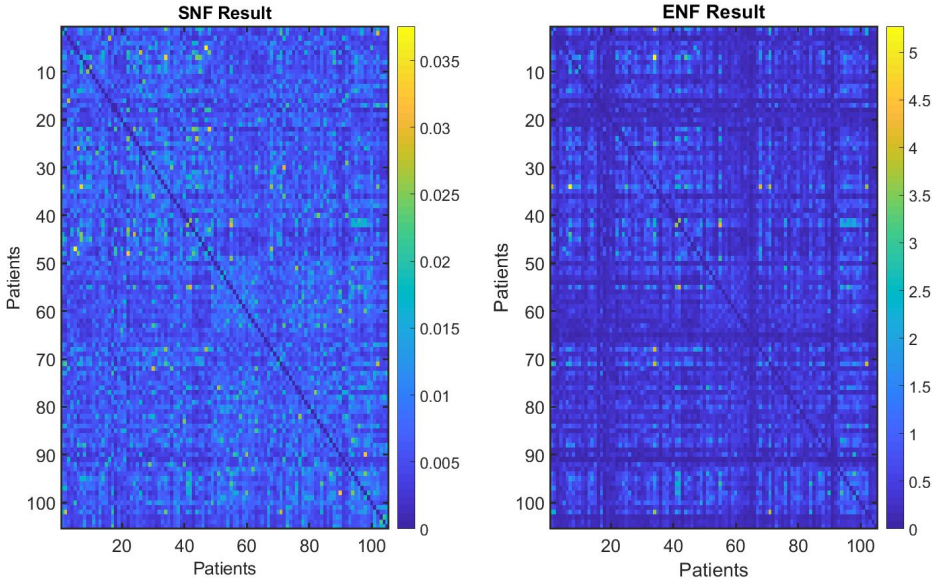

FIGURE 5.4: Heatmaps of the resulting network adjacency matrices obtained from SNF and ENF. Note that the diagonal was removed from both of these to improve the resolution for the SNF solution, as in the main text.

naturally then, by using eigengap, the number of clusters would be 2 and then 5 in that order. Eigengap looks at intervals between the eigenvalues and uses that to determine the number of clusters in the matrix best splits into [54]. In the following tables, Table 5.5, 5.6, 5.7 and 5.8, the first three columns have the method with the better value coloured green and the other one coloured red. This is to differentiate which method is better when comparing to each input layer. The last column contains the average of the highlighted values for each method and to the side these average values are compared and percentage improvements given. As can be seen, the average value for the distance between output and the inputs has the largest reduction, which is to be expected as this quantity is being minimised by our cost function. For the other values we see that whilst they are slightly worse for layers 1 and 3 there is significant improvement with the values for layer 2. Overall though the average values show that ENF generally outperform SNF.

As the above evaluations were done for a specific number of clusters, we give consideration to the possibility of the solutions being clustered into a different number of clusters. The average Concordance Index, NVI and NID were computed and plotted in Figure 5.5 when the output and the input networks were clustered into $k = 2, \ldots, 6$ components. This figure shows that for all clusters in this range, apart from $k = 4$, ENF produces better results across all of our quantities.

Next, we performed a survival analysis on the clustered results from ENF. The survival profiles for each of the clusters are shown in Figure 5.6 and their survival quartiles are shown in Table 5.9. We see that the Green and Blue group have a similar profile in that neither of them drop to 0%, and in fact they do not drop below 30%. This means that after 100 and 190 months respectively 30% of patients in those subgroups are either cured or survive past the end of the study. This may indicate subgroups that may be more easily treatable and they only differ by the points in which they start to decline. The Red and Yellow groups both experience rapid rates of decline, approximately 20 and 35 months for their IQR, differing in where the decline begins. This would indicate that the cancer in these subgroups that develop faster that the others. The last group, Orange, differs from the others by having the most steady rate of decline with an IQR of 103 months. The Log-Rank Test (LRT) was run on the clustered results from SNF and ENF and the resulting p-values shown in Table 5.10, both the SNF and the ENF value is below the 0.05 significance threshold. In fact, we also put the other clustering outputs of ENF from the range $[2, \ldots, 6]$ into the LRT and all the p-values obtained, except for $k = 3$ were below the 0.05 significance threshold, shown in Table 5.11.

$\sqrt{JS_{\mathcal{N}}}$	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer1	0	-	-	-	-	-	
Layer2	0.193	0	-	-	-	-	
Layer3	0.2096	0.2677	0	-	-	-	
SNF	0.106	0.1646	0.2423	0	-	0.171	
ENF	0.10	0.1404	0.1490	0.1244	0	0.1299	↓ 24.04%

TABLE 5.5: Shows the distance, $\sqrt{JS_{\mathcal{N}}}$, between each of the networks and the result from SNF and ENF. The last column denotes the average distance from the solutions to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

CI	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	1	-	-	-	-	-	
Layer 2	0.1594	1	-	-	-	-	
Layer 3	0.2316	0.0857	1	-	-	-	
SNF	0.4187	0.2546	0.3592	1	-	0.3442	
ENF	0.3633	0.4646	0.2655	0.6953	1	0.3645	↑ 5.9%

TABLE 5.6: Each of the networks, including the outputs, were clustered into five clusters and then the CI (agreement) between each of the clusterings were calculated. The last column denotes the average concordance between the solutions and the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

NVI	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	0	-	-	-	-	-	
Layer 2	0.9135	0	-	-	-	-	
Layer 3	0.8702	0.9553	0	-	-	-	
SNF	0.7372	0.8544	0.7811	0	-	0.7909	
ENF	0.7780	0.6983	0.8486	0.4731	0	0.7750	↓ 2%

TABLE 5.7: Each of the networks, including the outputs, were clustered into five clusters and the distance between each of the clusterings using the NVI metric was calculated. The last column is the average of the distance from the solution to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

NID	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	0	-	-	-	-	-	
Layer 2	0.8495	0	-	-	-	-	
Layer 3	0.7962	0.9201	0	-	-	-	
SNF	0.6251	0.7585	0.6470	0	-	0.6769	
ENF	0.6409	0.5664	0.7691	0.3846	0	0.6588	↓ 2.7%

TABLE 5.8: Each of the networks, including the outputs, were clustered into five clusters and the distance between each of the clusterings using the NID metric was calculated. The last column is the average of the distance from the solution to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

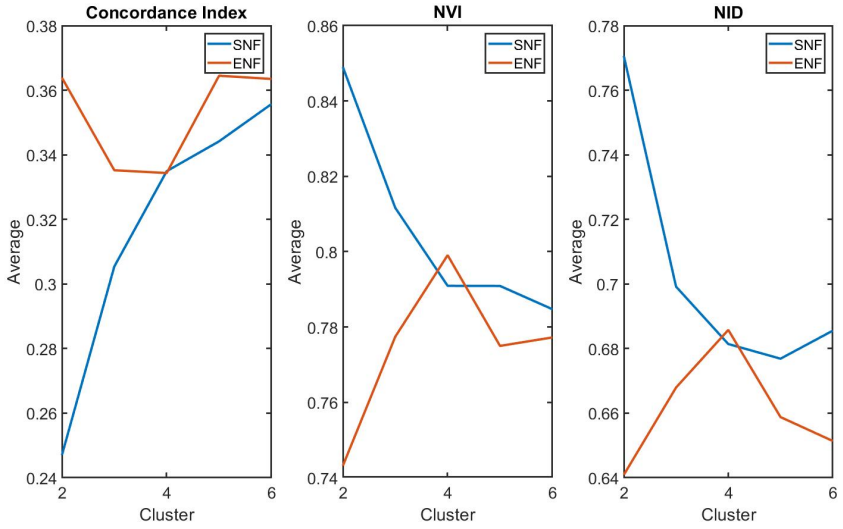

FIGURE 5.5: Tables 5.5 - 5.8 showed the values for CI, NVI and NID for the Breast Cancer dataset for a fixed number of clusters, namely five. Here we show the average values for these quantities for a range of clusters, two to six.

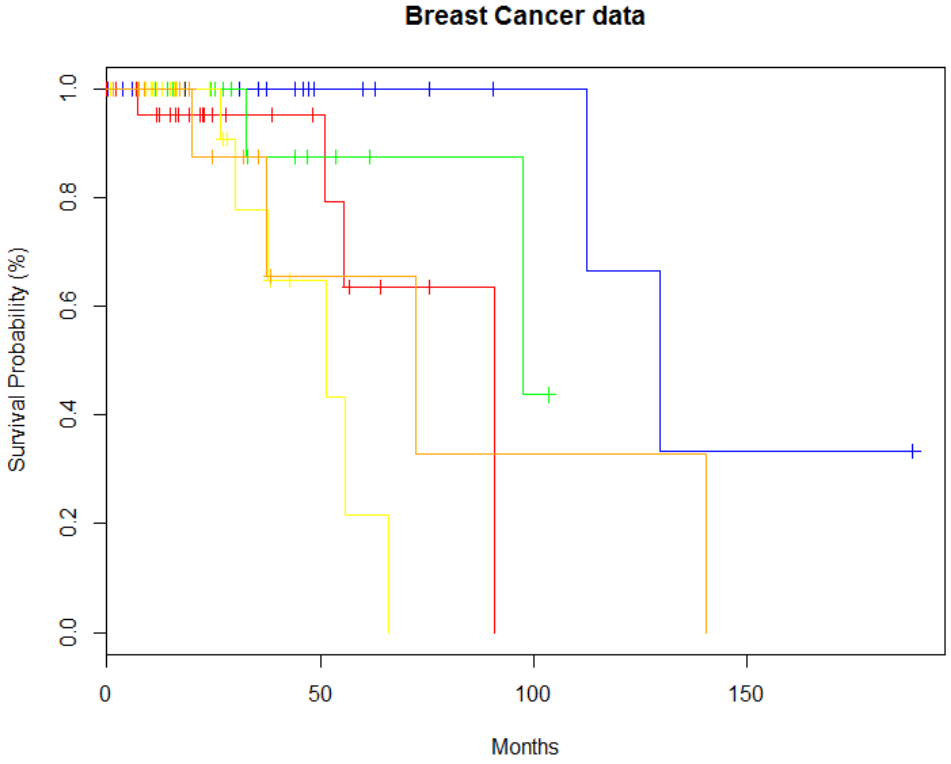

FIGURE 5.6: Kaplan-Meier survival curves for each of the five groups identified from the results of ENF applied to the Breast cancer data. Contains 5 cluster: Red - 25 patients; Blue - 20 patients; Green - 19 patients; Yellow - 25 patients; Orange - 16 patients.

Group	$\hat{t}_{25}$	$\hat{t}_{50}$	$\hat{t}_{75}$
Red	55.6	90.7	90.7
Blue	112.3	129.6	-
Green	97.4	97.4	-
Yellow	37.7	51.4	55.8
Orange	37.5	72.5	140.4

TABLE 5.9: This table shows the respective survival quartiles from the survival curves shown in Fig. 5.6.

Method	p-value
SNF	2.17×10^{-3}
ENF	4.5×10^{-3}

TABLE 5.10: Given the clustering result, (5 clusters), from both ENF and SNF on the Breast data, this table shows the p-values from the LRT.

Clusters	p-value
2	1.3×10^{-2}
3	0.271
4	4×10^{-4}
6	1.6×10^{-3}

TABLE 5.11: This tables shows the p-values from the LRT when the ENF solution is clustered into a different number of clusters.

5.2.2.2 Colon Cancer

As we have already stated, this subsection and subsequent subsection follow the same format as that of the previous section. To begin with, we show in Figure 5.7 the colour plot of the input networks used in both SNF and ENF. Note that like with the breast data layer 1 and 2 are on a much smaller scale, 10^{-5}, than the third layer, 10^{-3}. Again, we selected a random initialisation of θ rather than an average initialisation to prevent the skewed average from slowing down the convergence.

Next we have the outputs from the fusion procedure, starting with the cost history in Figure 5.8. The cost initialised at 0.0329, finishing at 0.0172, a reduction of 53.2% and the corresponding delta-change for this data set is $\delta = 1.5 \times 10^{-9}$. This and Figure 5.8 would indicate that the solution has converged. For comparison, the SNF solution achieves a cost of 0.0234 with respect to the ENF cost function. Once again, upon completion of the process, we embedded the output, its evolution and the input networks into 3 dimensions, shown in Figure 5.9. Here we again see the effect of the third layer skewing the average by their proximity, but also we see the method has converged on the at the barycenter, as expected. Lastly, Figure 5.10 shows the output of ENF side by side with SNF. Whilst the structure of both results seems similar, the structure of the ENF solution is clearer and more well-defined than that of the SNF solution.In terms of time taken for ENF to complete, using α equal to the number of patients, 92, and 7500 iterations, ENF took 4 minutes and 24 seconds to complete.

Now that we have the results from both SNF and ENF we evaluate them using our metrics. The evaluations requiring clustering used 3 clusters, matching that given in the SNF paper and clustered using spectral clustering. If the number of clusters were to be chosen according to the number of clusters that arose naturally then, by using eigengap [54], the number of clusters would be 3 and then 2 in that order. The following tables, Table 5.12, 5.13, 5.14 and 5.15, give the values, and average values, obtained for both solutions which we have coloured in the same fashion as before. Once again we see that for the distance between the networks, ENF has yielded a significant reduction, compared to SNF, but it is not unsurprising. As we can see the only value where ENF does not yield an improvement is with respect to the average NVI but this is an extremely small increase. The rest however are all better than those obtained from SNF. The reason why these values are so similar is that the ENF solution and the SNF are very similar which is shown by their small distance from each other shown in Table 5.12.

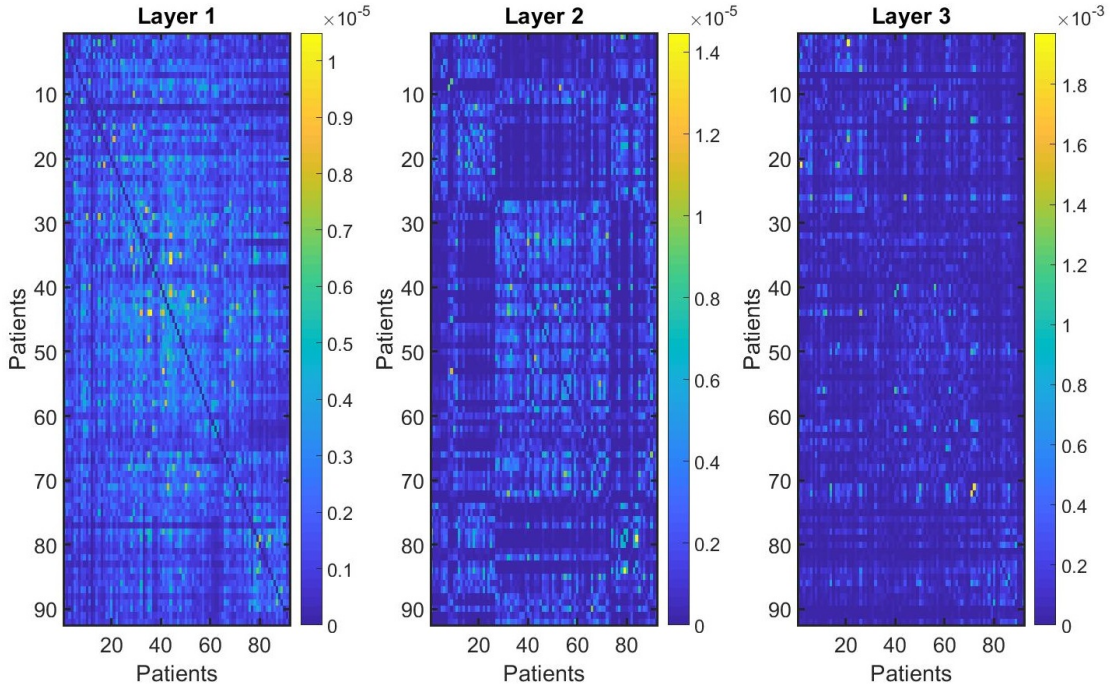

FIGURE 5.7: Heatmaps of the three adjacency matrices that were given as inputs to ENF for the Colon dataset. These initial inputs were created by the SNF method and so were given to ENF to keep our comparison as fair as possible. Layer 1 is mRNA expression, Layer2 is DNA methylation and Layer 3 is miRNA expression.

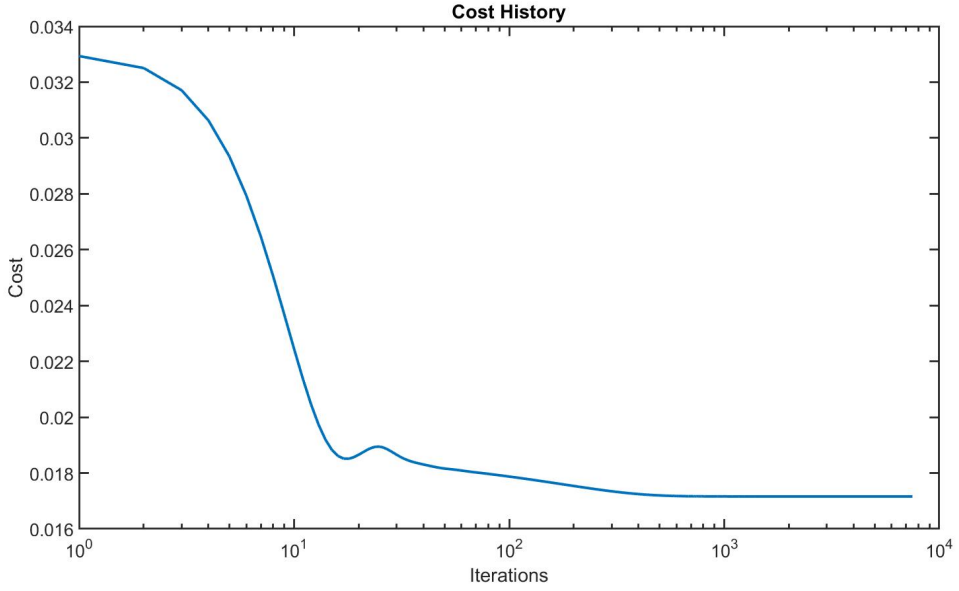

FIGURE 5.8: The value of the ENF cost function at each iterations of the integration process.

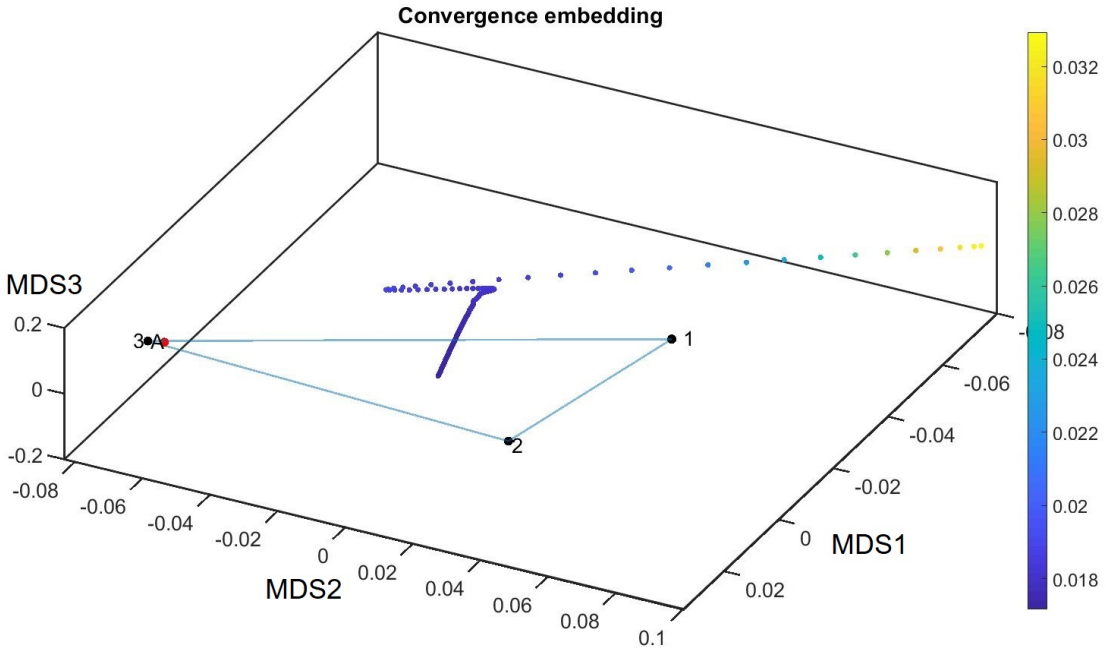

FIGURE 5.9: A visualisation of the networks embedded in $\mathbb{R}^3$ using MDS and the $\sqrt{JS_{\mathcal{N}}}$ as distances. The points labeled 1, 2 and 3 are the input networks shown in Figure 5.7; the point labeled 'A' is the average of the input networks; the remaining points are the complete history of theta coloured with respect to the cost at that time, converging to the (approximate) barycenter, as expected.

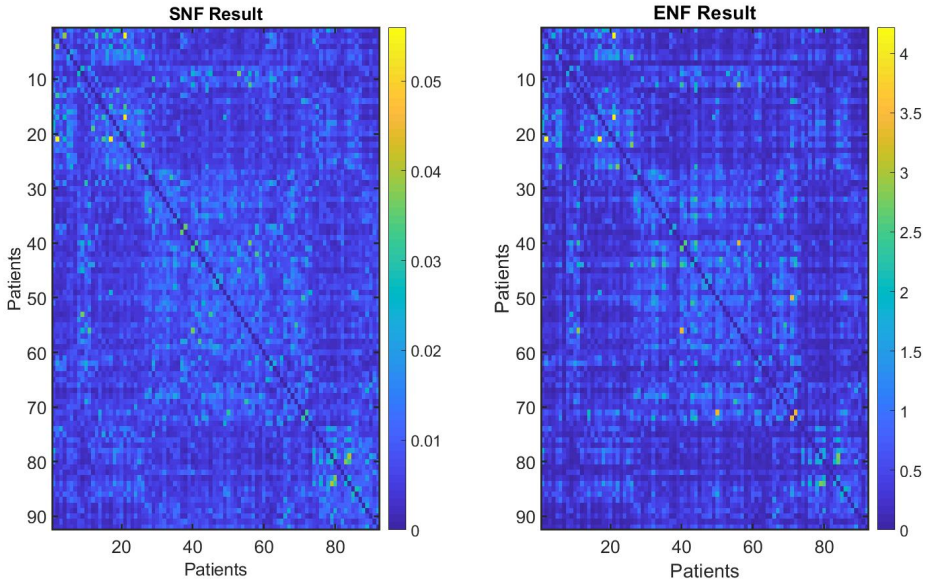

FIGURE 5.10: Heatmaps of the resulting network adjacency matrices obtained from SNF and ENF. Note that the diagonal was removed from both of these, as explained in the Breast cancer section, to improve the resolution for the SNF solution.

Again, we give consideration to the effect of the number of clusters on the values of our metrics. The average values are calculated for when the output and the input networks were clustered into $k = 2, \ldots, 6$ components and plotted in Figure 5.11. We see that only for NVI, when $k = 3$, SNF beats ENF, which is extremely close. Apart from that, all other values yield a good improvement. This shows that even though the results from SNF and ENF are similar for this data set, ENF picks up subtler connections, producing an overall better result than SNF.

The survival profiles for each of the clusters are shown in Figure 5.12. In this case the three profiles are very distinct. Blue, containing 23% of the patients, has a 100% survival rate; Green appears to have a consistent survival rate up untill a critical point, month 39, where it significantly decreases; Red however has a poor initial start with only 50% of the groups living beyond 15 months, but stabilising after that. The survival quartiles can be found in Table 5.16. The LRT was run on the clustered results from SNF and ENF and the resulting p-values shown in Table 5.17, in this case only the ENF value is below the 0.05 significance threshold, whereas the SNF solution is not, meaning that there are significant difference between survival curves of the clustering found from the ENF output. In fact, we also put the other clustering outputs of ENF from the range $[2, \ldots, 6]$ into the LRT and all the p-values obtained, except for $k = 6$, were below the 0.05 significance threshold, shown in Table 5.18.

$\sqrt{JS_{\mathcal{N}}}$	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer1	0	-	-	-	-	-	
Layer2	0.1841	0	-	-	-	-	
Layer3	0.2338	0.2497	0	-	-	-	
SNF	0.1098	0.1429	0.194	0	-	0.1489	
ENF	0.1141	0.1254	0.1508	0.0799	0	0.1301	↓ 12.6%

TABLE 5.12: Shows the distance, $\sqrt{JS_{\mathcal{N}}}$, between each of the networks and the result from SNF and ENF. The last column denotes the average distance from the solutions to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

CI	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	1	-	-	-	-	-	
Layer 2	0.0129	1	-	-	-	-	
Layer 3	0.0309	0.076	1	-	-	-	
SNF	0.2949	0.3932	0.0724	1	-	0.2535	
ENF	0.2884	0.3594	0.1252	0.7435	1	0.2577	↑ 1.7%

TABLE 5.13: Each of the networks, including the outputs, were clustered into three clusters and then the CI (agreement) between each of the clusterings were calculated. The last column denotes the average concordance between the solutions and the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

NVI	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	0	-	-	-	-	-	
Layer 2	0.9935	0	-	-	-	-	
Layer 3	0.9844	0.9606	0	-	-	-	
SNF	0.8272	0.7555	0.9624	0	-	0.8484	
ENF	0.8315	0.7809	0.9333	0.4087	0	0.8486	↑ 0.02%

TABLE 5.14: Each of the networks, including the outputs, were clustered into three clusters and the distance between each of the clusterings using the NVI metric was calculated. The last column is the average of the distance from the solution to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

NID	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	0	-	-	-	-	-	
Layer 2	0.9871	0	-	-	-	-	
Layer 3	0.971	0.9284	0	-	-	-	
SNF	0.7165	0.6208	0.9293	0	-	0.7555	
ENF	0.7138	0.6422	0.8815	0.2797	0	0.7458	↓ 1.28%

TABLE 5.15: Each of the networks, including the outputs, were clustered into three clusters and the distance between each of the clusterings using the NID metric was calculated. The last column is the average of the distance from the solution to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

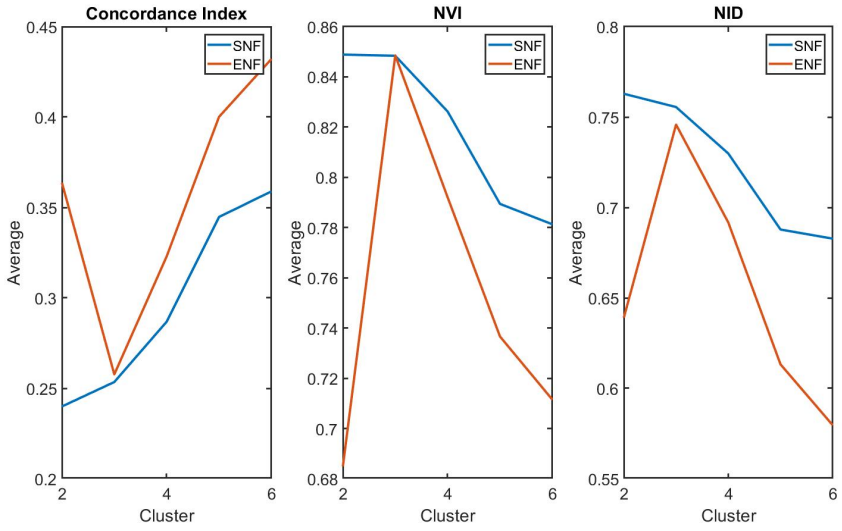

FIGURE 5.11: Tables 5.12 - 5.15 showed the values for CI, NVI and NID for the Colon Cancer data set for a fixed number of clusters, namely three. Here we show the average values for these quantities for a range of clusters, two to six.

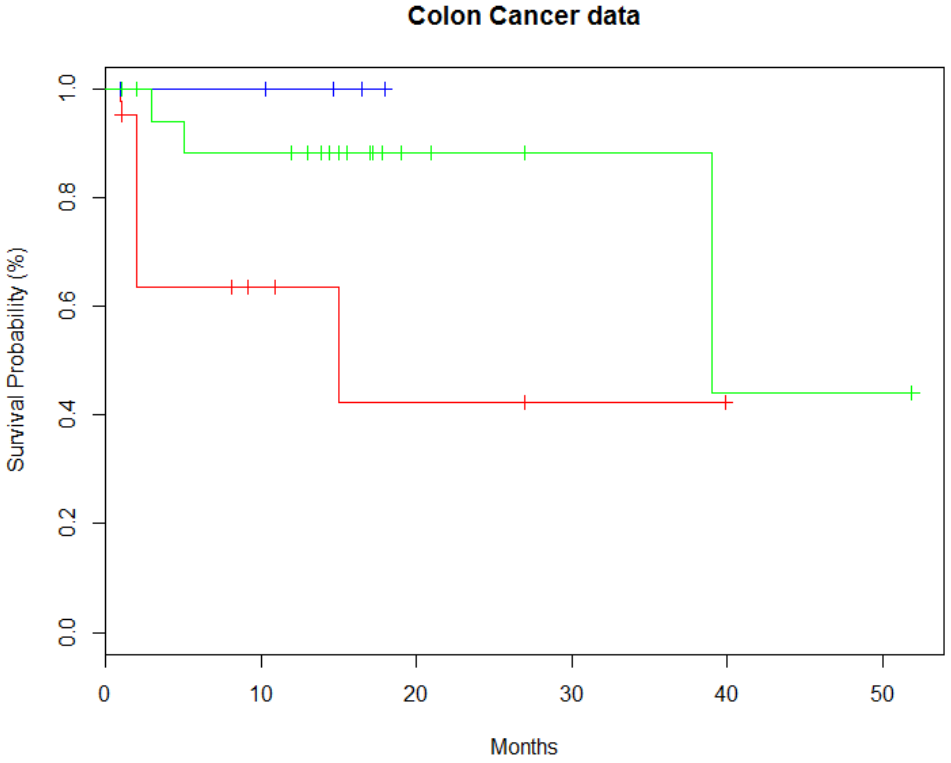

FIGURE 5.12: Kaplan-Meier survival curves for each of the three groups identified from the results of ENF applied to the Colon cancer data. Contains 3 cluster: Red - 44 patients; Blue - 21 patients; Green - 27 patients.

Group	$\hat{t}_{25}$	$\hat{t}_{50}$	$\hat{t}_{75}$
Red	2	15	-
Blue	-	-	-
Green	39	39	-

TABLE 5.16: This table shows the respective survival quartiles from the survival curves shown in Figure 5.12

Method	p-Value
SNF	8.21×10^{-2}
ENF	4.38×10^{-2}

TABLE 5.17: Given the clustering result, (3 clusters), from both ENF and SNF on the Colon data, this table shows the p-values from the LRT.

Clusters	p-value
2	2.6×10^{-2}
4	1.2×10^{-2}
5	2.7×10^{-2}
6	0.159

TABLE 5.18: This table shows the p-values from the LRT when the ENF solution is clustered into a different number of clusters.

5.2.2.3 Glio Cancer

Starting with the inputs, Figure 5.13 shows the colour plot of the input networks used in both SNF and ENF, which has the same scale issue as the other data set and so a random initialisation of θ implemented. The cost history from the ENF procedure is shown in Figure 5.14. Starting at 0.0521 and finishing at 0.0279 this yields a reduction of 46.5%. The SNF solution achieves a cost of 0.0415 in the ENF cost function. For this data set the delta-change is $\delta = 3.5 \times 10^{-11}$. This and the cost history shown in Figure 5.14 indicates that this has converged. Once completed the networks were embedded into 3 dimensions, in the same fashion as before which is shown in Figure 5.15. Following this, Figure 5.16 shows the output of ENF side by side with SNF. Whilst the clusters are not so clear and well defined we do see more structure in the ENF solution than the SNF solution. In terms of time taken for ENF to complete, using α equal to the number of patients, 215, and 15000 iterations, ENF took 1 hour, 43 minutes and 37 seconds to complete. This large increase in time is due to the large number of patients and how the complexity of eigenvalues grow.

Now we apply our metrics to the results from ENF and SNF. For the evaluations requiring clusterings, 3 clusters were used matching that given in the SNF paper. If the number of clusters were to be chosen according the number of clusters that arose naturally then, by using eigengap [54], the number of clusters would be 2 and then 4 in that order. The following tables, Table 5.19, 5.20, 5.21 and 5.22, give the values obtained for both solutions and coloured in the same fashion as before. In this data set we have a different situation. The only value where ENF does yield an improvement is with respect to the QJSD, whilst the rest are worse by a very small amount. When we consider other number of clusters, shown and plotted in Figure 5.17, we see that the values for SNF and ENF are extremely close and similar. This indicates thats for for this data set, the ENF and SNF solution are of similar quality.

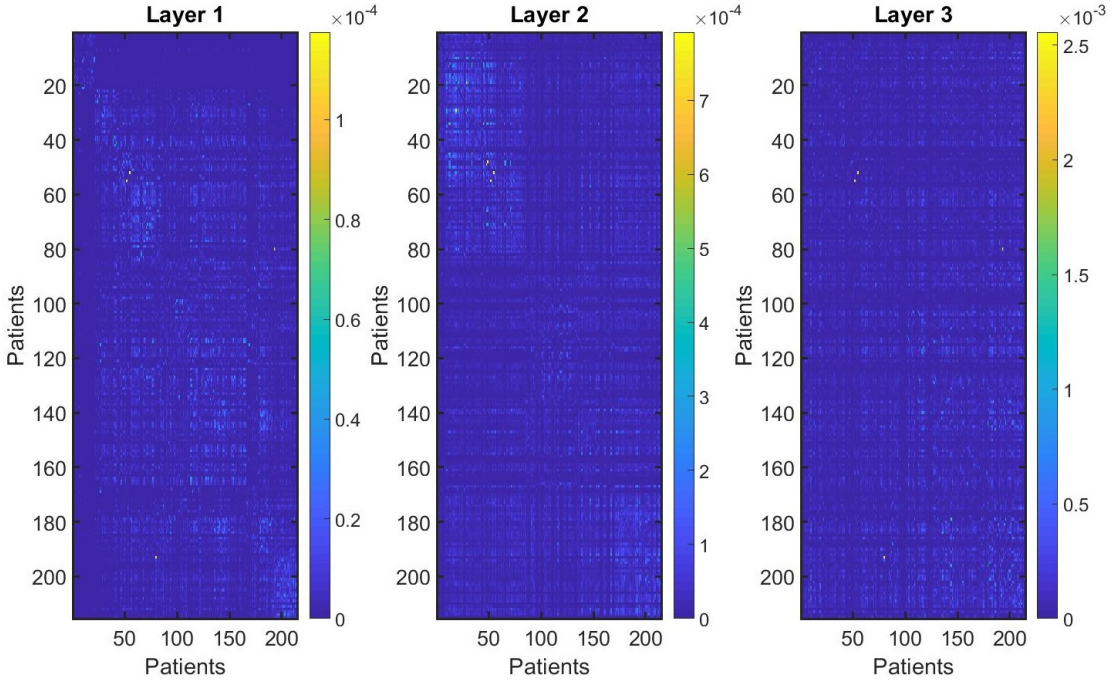

FIGURE 5.13: Heatmaps of the three adjacency matrices that were given as inputs to ENF for the Glio dataset. These initial inputs were created by the SNF method and so were given to ENF to keep our comparison as fair as possible. Layer 1 is mRNA expression, Layer2 is DNA methylation and Layer 3 is miRNA expression.

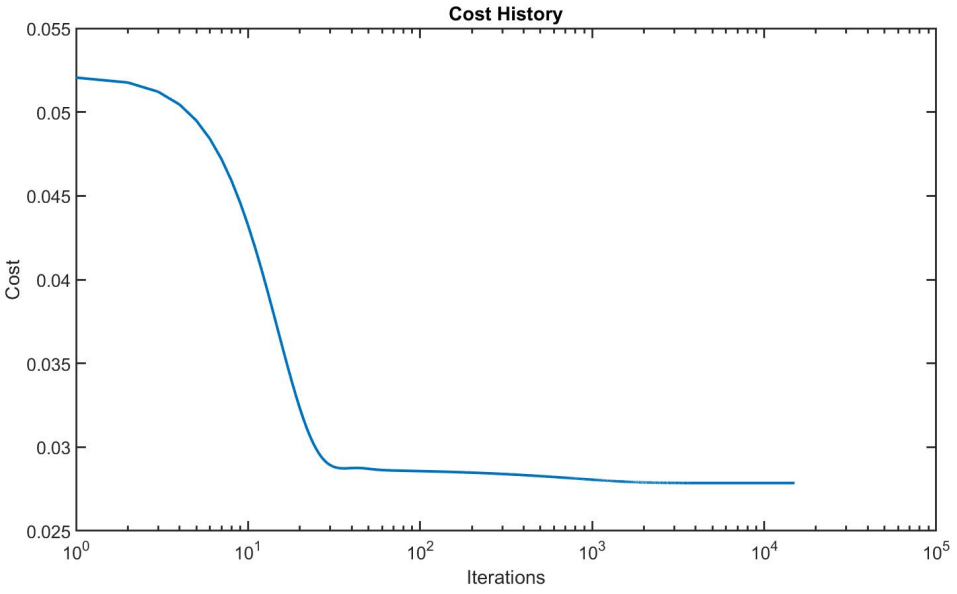

FIGURE 5.14: The value of the ENF cost function at each iterations of the integration process.

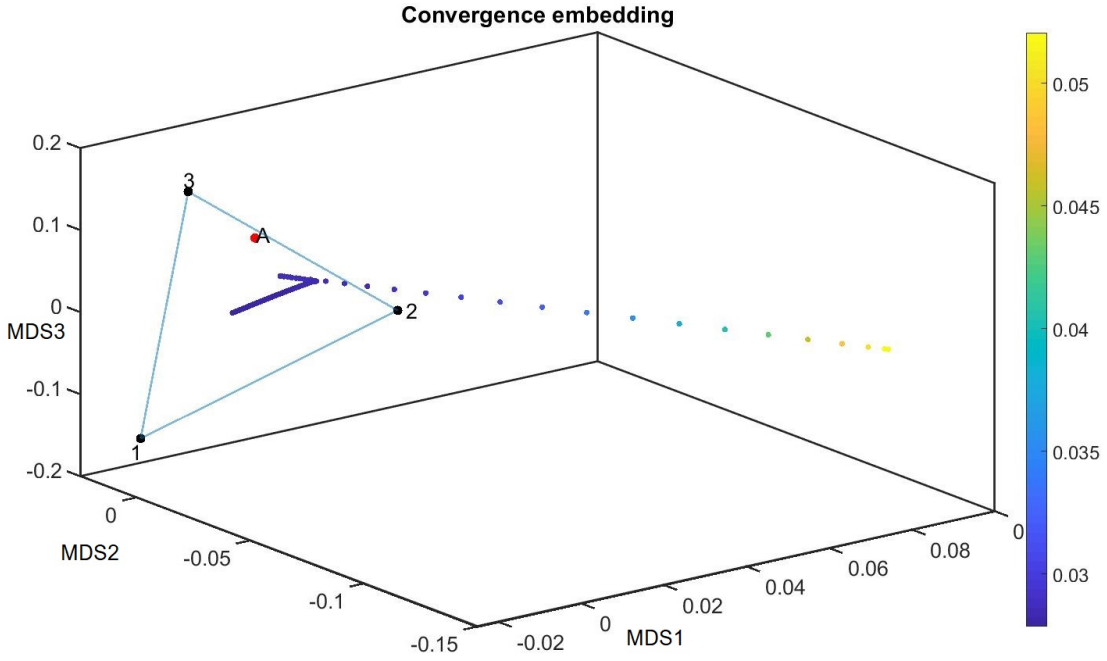

FIGURE 5.15: A visualisation of the networks embedded in $\mathbb{R}^2$ using MDS and the $\sqrt{JS_{\mathcal{N}}}$ as distances. The points labeled 1, 2 and 3 are the input networks shown in Figure 5.13; the point labeled 'A' is the average of the input networks; the remaining points are every other point in the history of theta and coloured with respect to the cost at that time. Again it appears to have converged at the barycenter.

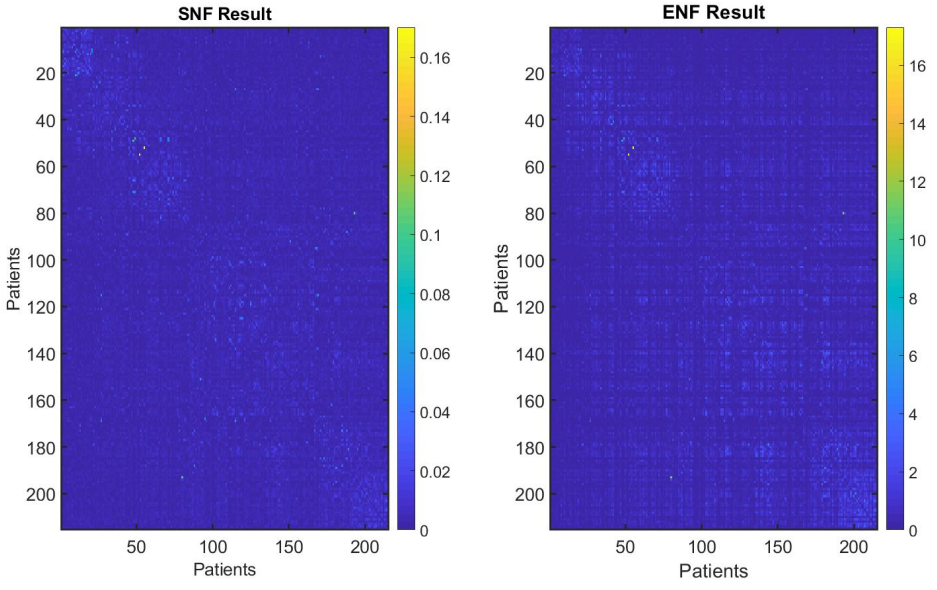

FIGURE 5.16: Heatmaps of the resulting network adjacency matrices obtained from SNF and ENF. Note that the diagonal was removed from both of these, as explained in the Breast cancer section, to improve the resolution for the SNF solution.

The survival profiles for each of the clusters are shown in Figure 5.18 and its survival percentiles in Table 5.23. In this it can be seen that the Green group has better survival prospects with an IQR of 37 month with 50% of surving beyond 18 months. This highlights the poorer prospects of the Red group as only 25% survive beyond 20.3 months. The Blue group falls between the other two with a similar start to Red, but after month 20 has better a stronger profile. We ran the LRT on the clustered results from SNF and ENF and the resulting p-values are shown in Table 5.24, both the ENF and SNF are below the 0.05 significance threshold. In fact, we also put the other clustering outputs of ENF from the range $[2, \ldots, 6]$ into the LRT and all the p-values obtained, except for $k = 2$, were below the 0.05 significance threshold, shown in Table 5.25.

$\sqrt{JS_{\mathcal{N}}}$	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer1	0	-	-	-	-	-	
Layer2	0.2916	0	-	-	-	-	
Layer3	0.2769	0.2838	0	-	-	-	
SNF	0.2142	0.1888	0.2076	0	-	0.2035	
ENF	0.1677	0.1702	0.1627	0.1192	0	0.1669	↓ 18%

TABLE 5.19: Shows the distance, $\sqrt{JS_{\mathcal{N}}}$, between each of the networks and the result from SNF and ENF. The last column denotes the average distance from the solutions to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

CI	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	1	-	-	-	-	-	
Layer 2	0.0733	1	-	-	-	-	
Layer 3	0.0124	0.0629	1	-	-	-	
SNF	0.1457	0.1838	0.1478	1	-	0.1591	
ENF	0.1484	0.1327	0.1793	0.6276	1	0.1535	↓ 3.55%

TABLE 5.20: Each of the networks, including the outputs, were clustered into three clusters and then the CI (agreement) between each of the clusterings were calculated. The last column denotes the average concordance between the solutions and the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

NVI	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	0	-	-	-	-	-	
Layer 2	0.9635	0	-	-	-	-	
Layer 3	0.9938	0.9685	0	-	-	-	
SNF	0.9215	0.9027	0.9202	0	-	0.9148	
ENF	0.9199	0.9325	0.9018	0.5434	0	0.9181	↑ 0.4%

TABLE 5.21: Each of the networks, including the outputs, were clustered into three clusters and the distance between each of the clusterings using the NVI metric was calculated. The last column is the average of the distance from the solution to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

NID	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	0	-	-	-	-	-	
Layer 2	0.9449	0	-	-	-	-	
Layer 3	0.9881	0.9507	0	-	-	-	
SNF	0.8567	0.8595	0.8558	0	-	0.8574	
ENF	0.8558	0.9031	0.833	0.4005	0	0.8640	↑ 0.8%

TABLE 5.22: Each of the networks, including the outputs, were clustered into three clusters and the distance between each of the clusterings using the NID metric was calculated. The last column is the average of the distance from the solution to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

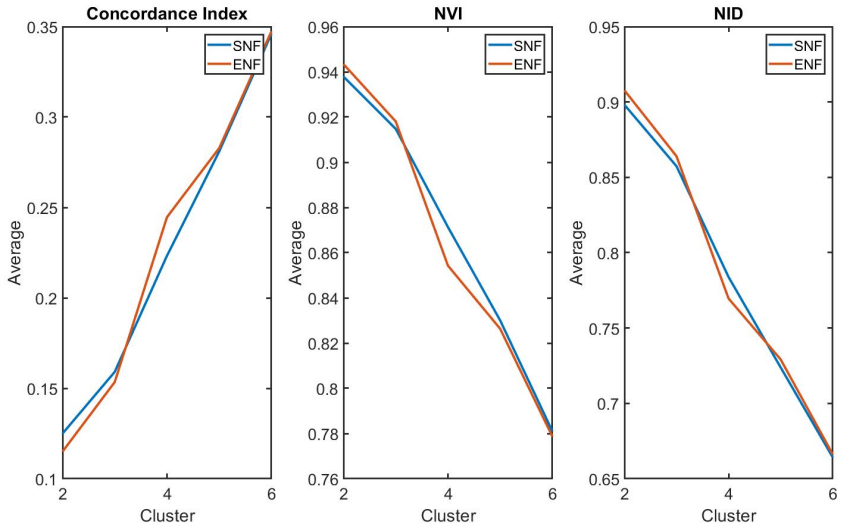

FIGURE 5.17: Tables 5.19 - 5.22 showed the values for CI, NVI and NID for the Glio Cancer data set for a fixed number of clusters, namely three. Here we show the average values for these quantities for a range of clusters, two to six.

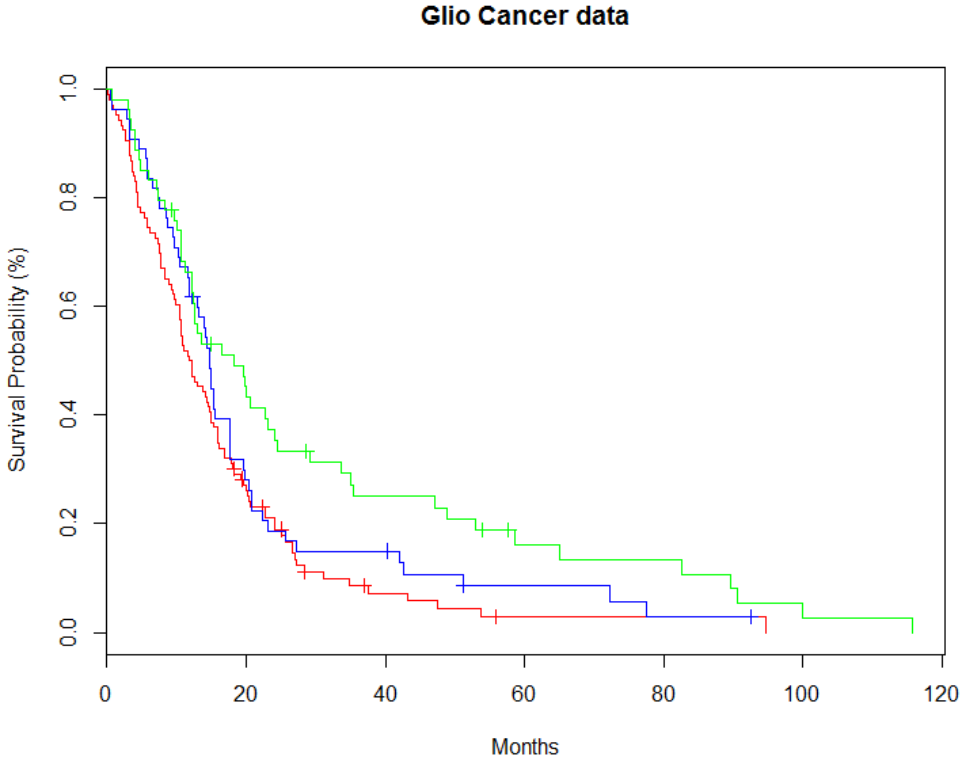

FIGURE 5.18: Kaplan-Meier survival curves for each of the three groups identified from the results of ENF applied to the Glio cancer data. Contains 3 cluster: Red - 106 patients; Blue - 55 patients; Green - 54 patients.

Group	$\hat{t}_{25}$	$\hat{t}_{50}$	$\hat{t}_{75}$
Red	5.8	12	20.3
Blue	8.8	14.7	20.8
Green	10.2	18.3	47.1

TABLE 5.23: This table shows the respective survival quartiles from the survival curves shown in Figure 5.18.

Method	p-Value
SNF	6.31×10^{-4}
ENF	9.5×10^{-3}

TABLE 5.24: Given the clustering result, (3 clusters), from both ENF and SNF on the Glio data, this table shows the p-values from the LRT.

Clusters	p-value
2	0.49
4	3.9×10^{-3}
5	7.4×10^{-3}
6	8.6×10^{-3}

TABLE 5.25: This tables shows the p-values from the LRT when the ENF solution is clustered into a different number of clusters.

5.2.2.4 Kidney Cancer

To begin with Figure 5.19 shows the colour plot of the input networks used in both SNF and ENF, which has the same scale issue as the other data set and so a random initialisation of θ was implemented. The cost history, plotted in Figure 5.20, starts at 0.0484 and finishes at 0.017 giving a reduction of 64.8%. The SNF solution achieves a cost of 0.0325 in the ENF cost function. For this data set the delta-change is $\delta = -1.5\times10^{-11}$, given how small this value is and Figure 5.20 this would indicate it has converged. Once completed, we embedded the networks and the evolution of the solution into 3 dimensions in the same fashion as before, shown in Figure 5.21. Following this, Figure 5.22 shows the output of ENF side by side with SNF where the structure of the ENF solution is still clearer and more well defined. In terms of time taken for ENF to complete, using α equal to the number of patients, 122, and 10000 iterations, ENF took 11 minutes and 50 seconds to complete.

Following we have the various metrics evaluated for this data set, the evaluations requiring clustering used 3 clusters, matching that given in the SNF paper. If the number of clusters were to be chosen according the number of clusters that arose naturally then, by using eigengap [54], the number of clusters would be 5 and then 2 in that order. Tables 5.26, 5.27, 5.28 and 5.29, give the values obtained both solutions and coloured in the same fashion as before. Here we see significant improvements across all of our values. Particularly for CI, whilst SNF agrees more with the third layer we see at least double the amount of agreement with both the first and second layers. Even when we extend the number of clusters and plot the average values, Figure 5.23, we see that for all k in the given range the ENF values are much better than the SNF results. Overall this indicates a much better solution from ENF.

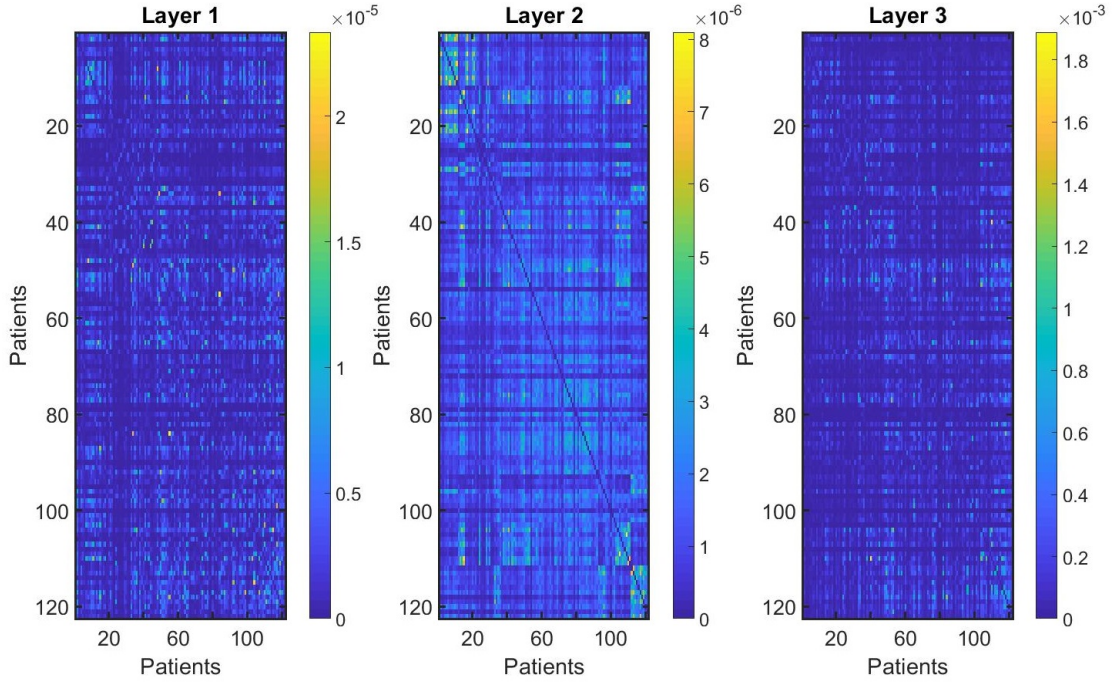

FIGURE 5.19: Heatmaps of the three adjacency matrices that were given as inputs to ENF for the Kidney dataset. These initial inputs were created by the SNF method and so were given to ENF to keep our comparison as fair as possible. Layer 1 is mRNA expression, Layer2 is DNA methylation and Layer 3 is miRNA expression.

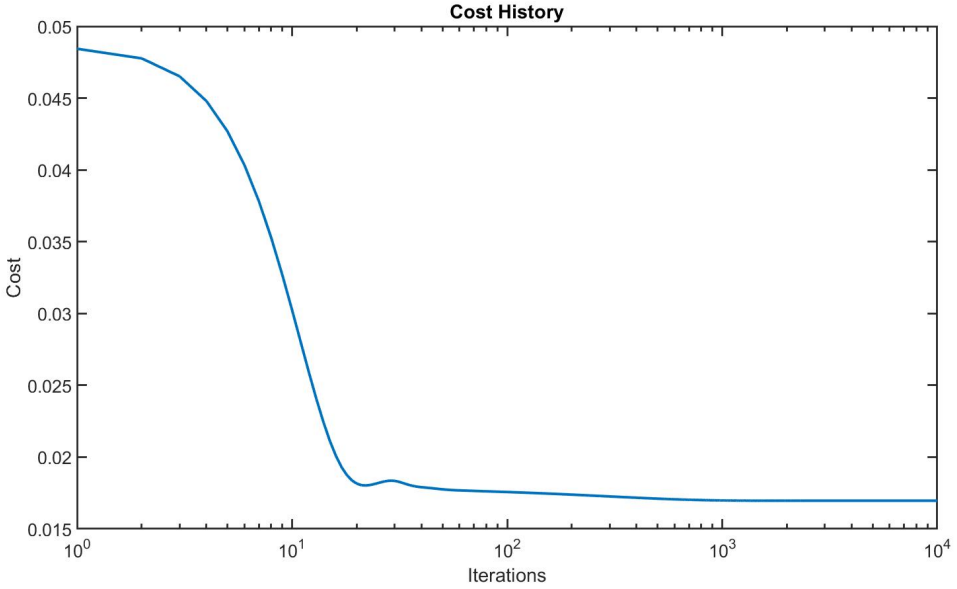

FIGURE 5.20: The value of the ENF cost function at each iterations of the integration process.

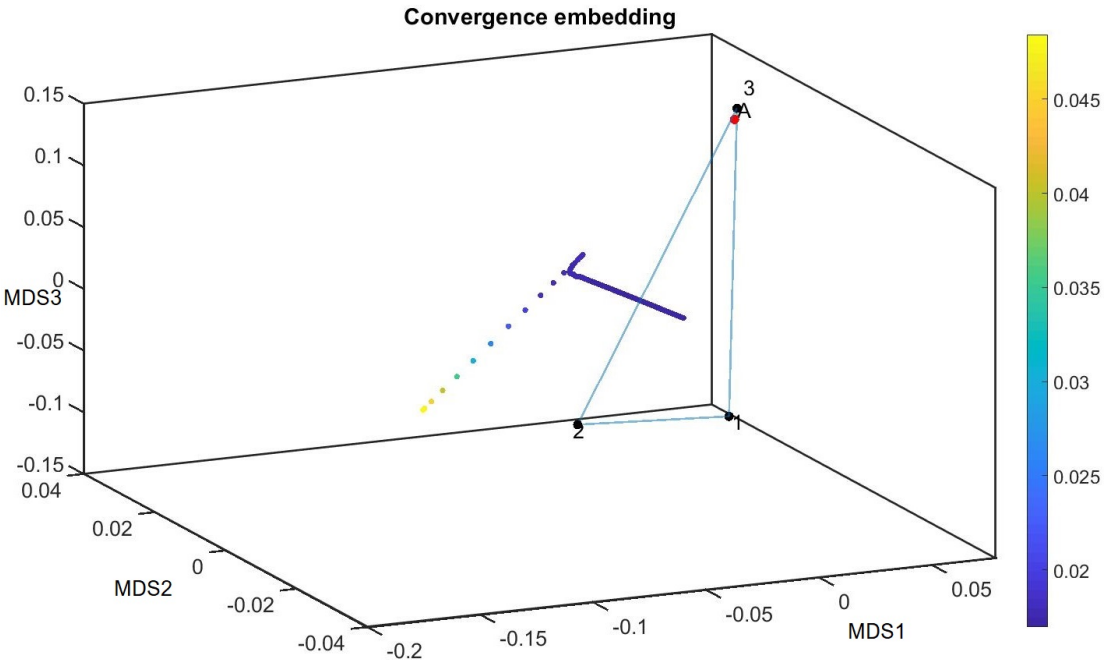

FIGURE 5.21: A visualisation of the networks embedded in $\mathbb{R}^3$ using MDS and the $\sqrt{JS_{\mathcal{N}}}$ as distances. The points labeled 1, 2 and 3 are the input networks shown in Figure 5.19; the point labeled 'A' is the average of the input networks; the remaining points are every other point in the history of theta and coloured with respect to the cost at that time. As can be seen it has reached the barycenter.

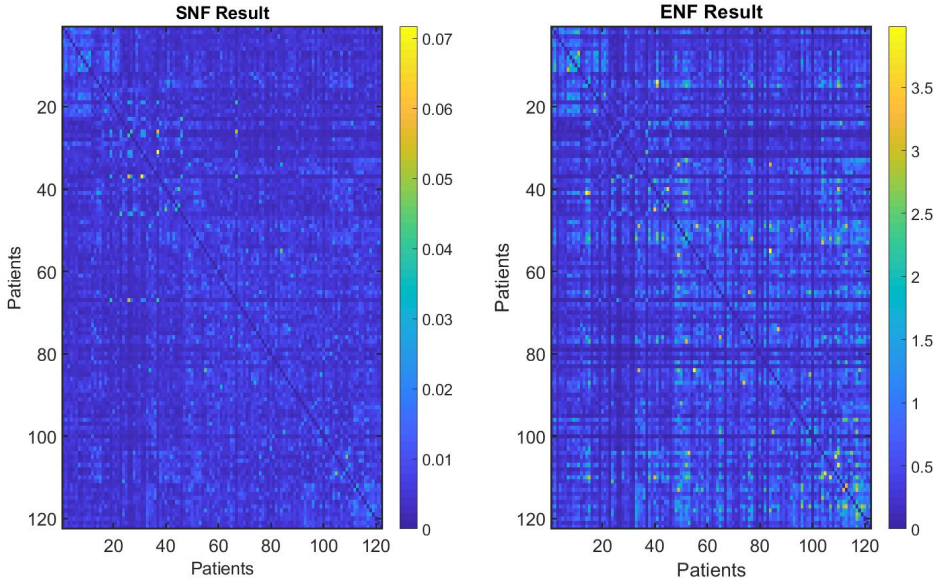

FIGURE 5.22: Heatmaps of the resulting network adjacency matrices obtained from SNF and ENF. Note that the diagonal was removed from both of these, as explained in the Breast cancer section, to improve the resolution for the SNF solution.

The survival profiles for each of the clusters are shown in Figure 5.24 and its survival percentiles in Table 5.30. Again we see there are 3 distinct profiles, with Blue only having a death and then no more; Red after suffering an initial decline stabilises at a constant rate and finally Green with a constant rate of decline. The LRT was run on the clustered results from SNF and ENF and the resulting p-values shown in Table 5.31, only the ENF is below the 0.05 significance threshold whilst SNF is not. This means that these clusters have distinct survival profiles that has not been attained by chance, showing the effectiveness of ENF. When we also put the other clustering outputs of ENF from the range $[2, \ldots, 6]$ into the LRT, this time none of the other p-values obtained were below the 0.05 significance threshold, shown in Table 5.32.

$\sqrt{JS_{\mathcal{N}}}$	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer1	0	-	-	-	-	-	
Layer2	0.2031	0	-	-	-	-	
Layer3	0.2258	0.2388	0	-	-	-	
SNF	0.1892	0.1196	0.2180	0	-	0.1756	
ENF	0.1201	0.1286	0.1411	0.1264	0	0.1299	↓ 26%

TABLE 5.26: Shows the distance, $\sqrt{JS_{\mathcal{N}}}$, between each of the networks and the result from SNF and ENF. The last column denotes the average distance from the solutions to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

CI	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	1	-	-	-	-	-	
Layer 2	0.0266	1	-	-	-	-	
Layer 3	0.0628	0.0512	1	-	-	-	
SNF	0.1092	0.1209	0.3536	1	-	0.1946	
ENF	0.3197	0.2282	0.2634	0.3727	1	0.2704	↑ 39%

TABLE 5.27: Each of the networks, including the outputs, were clustered into three clusters and then the CI (agreement) between each of the clusterings were calculated. The last column denotes the average concordance between the solutions and the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

NVI	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	0	-	-	-	-	-	
Layer 2	0.9865	0	-	-	-	-	
Layer 3	0.9676	0.9737	0	-	-	-	
SNF	0.946	0.9385	0.7963	0	-	0.8936	
ENF	0.8104	0.8712	0.8483	0.7818	0	0.8433	↓ 5.6%

TABLE 5.28: Each of the networks, including the outputs, were clustered into three clusters and the distance between each of the clusterings using the NVI metric was calculated. The last column is the average of the distance from the solution to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

NID	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	0	-	-	-	-	-	
Layer 2	0.9751	0	-	-	-	-	
Layer 3	0.9408	0.9493	0	-	-	-	
SNF	0.9237	0.9096	0.7377	0	-	0.8570	
ENF	0.7033	0.7733	0.7406	0.7194	0	0.7390	↓ 13.77%

TABLE 5.29: Each of the networks, including the outputs, were clustered into three clusters and the distance between each of the clusterings using the NID metric was calculated. The last column is the average of the distance from the solution to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

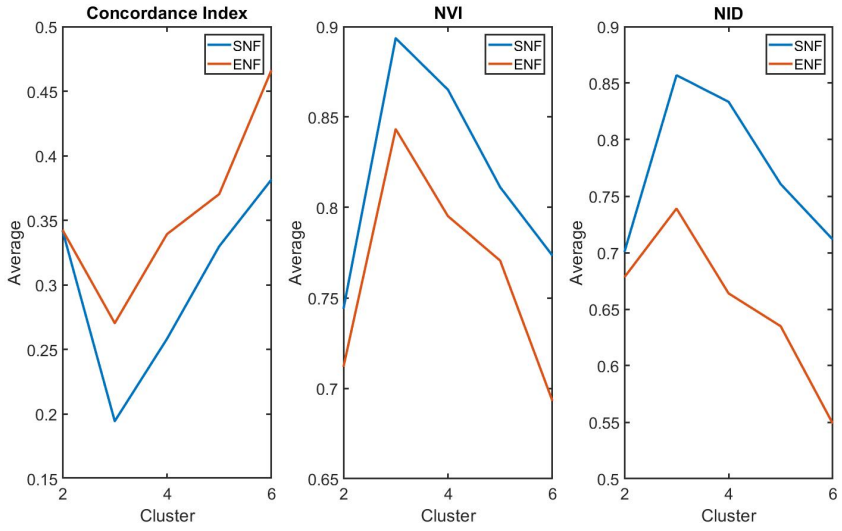

FIGURE 5.23: Tables 5.26 - 5.29 showed the values for CI, NVI and NID for the Kidney Cancer dataset for a fixed number of clusters, namely three. Here we show the average values for these quantities for a range of clusters, two to six.

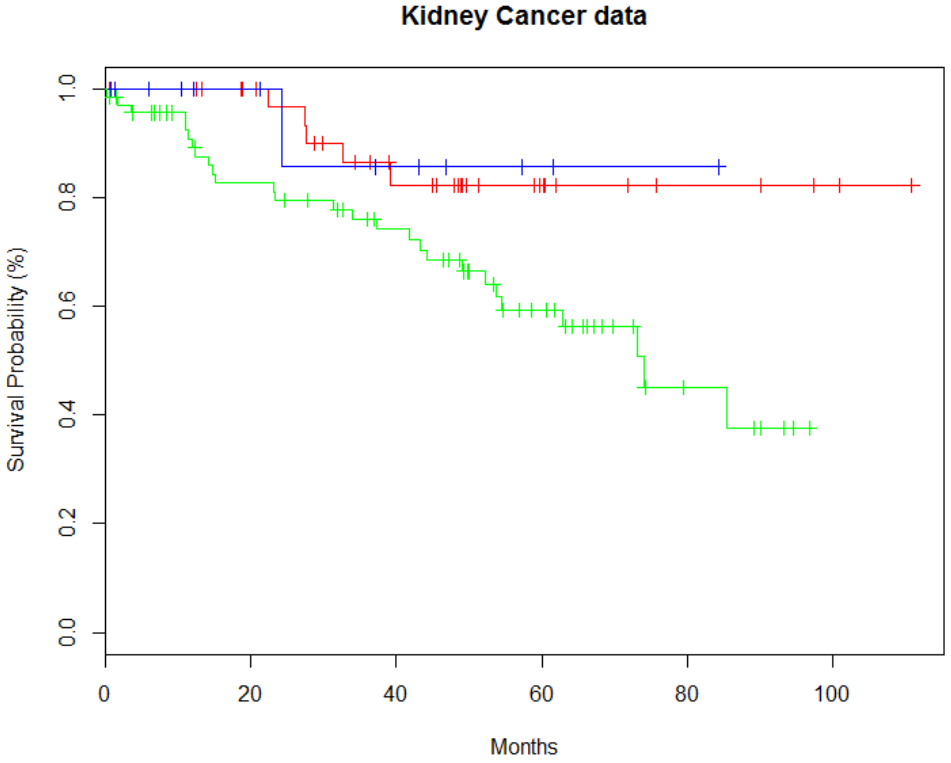

FIGURE 5.24: Kaplan-Meier survival curves for each of the three groups identified from the results of ENF applied to the Kidney cancer data. Contains 3 cluster: Red - 38 patients; Blue - 13 patients; Green - 71 patients.

Group	$\hat{t}_{25}$	$\hat{t}_{50}$	$\hat{t}_{75}$
Red	-	-	-
Blue	-	-	-
Green	37.2	74.1	-

TABLE 5.30: This table shows the respective survival quartiles from the survival curves shown in Figure 5.24.

Method	p-Value
SNF	0.138
ENF	3.52×10^{-2}

TABLE 5.31: Given the clustering result, (3 clusters), from both ENF and SNF on the Kidney data, this table shows the p-values from the LRT.

Clusters	p-value
2	0.844
4	0.147
5	0.104
6	0.211

TABLE 5.32: This tables shows the p-values from the LRT when the ENF solution is clustered into a different number of clusters.

5.2.2.5 Lung Cancer

Starting with the inputs for both fusions methods, Figure 5.25 shows the colour plot of the input networks. As usual its still has the same scale issue as the other data set and so a random initialisation of θ was implemented. The plot of the cost history is shown in Figure 5.26 where it starts at 0.0546 and finishes at 0.0208, reducing by 62%. The SNF solution achieves a cost of 0.0371 in the ENF cost function. For this data set the delta-change is $\delta = 5.8 \times 10^{-9}$, indicating convergence. Once completed, the networks and the evolution of the solution were embedded into 3 dimensions, shown in Figure 5.27. Where we still see that the average graph is closest to the third layer due to its scale. Following this, Figure 5.28 shows the output of ENF side by side with SNF where the structure of the ENF solution has more contrast and is more well defined than SNF. In terms of time taken for ENF to complete, using α equal to the number of patients, 106, and 7500 iterations, ENF took 6 minutes and 35 seconds to complete.

Following that we have the various metrics evaluated for this data set, the evaluations requiring clustering used 4 clusters, matching that given in the SNF paper. If the number of clusters were to be chosen according the number of clusters that arose naturally then, by using eigengap [54], the number of clusters would be 3 and then 4 in that order. Tables 5.33, 5.34, 5.35 and 5.36, give the values obtained for both solutions and coloured in the same fashion as before. These values show that overall the ENF solution is consistently better than the SNF solution. Upon closer inspection we see that whilst all the values for SNF are better with the second layer, indicating that SNF may be focusing on that layer more than the others. Meanwhile ENF give a more balanced attention to all layers, which is why we see the improvement in the other layers, particularly layer 1, where the biggest improvement can be seen. When the number of cluster changed to extended to run over a larger range and plot the average values we attain Figure 5.29. Whilst for 6 clusters the answer is unclear, this shows that for more than $3, 4, 5$ clusters that the ENF solution is still better than the SNF solution.

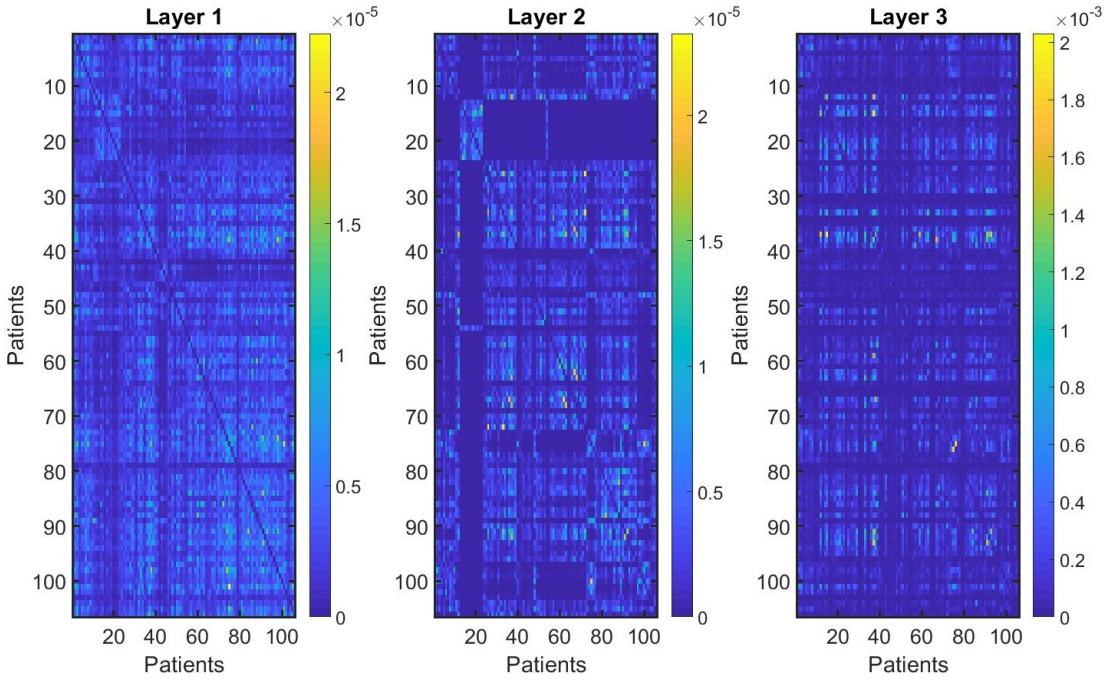

FIGURE 5.25: Heatmaps of the three adjacency matrices that were given as inputs to ENF for the Lung dataset. These initial inputs were created by the SNF method and so were given to ENF to keep our comparison as fair as possible. Layer 1 is mRNA expression, Layer2 is DNA methylation and Layer 3 is miRNA expression.

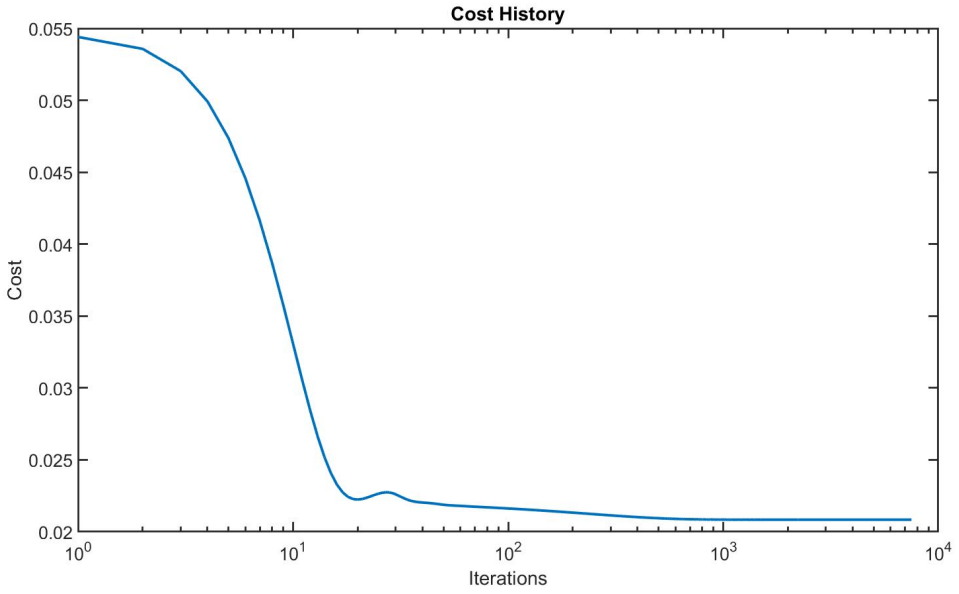

FIGURE 5.26: The value of the ENF cost function at each iterations of the integration process.

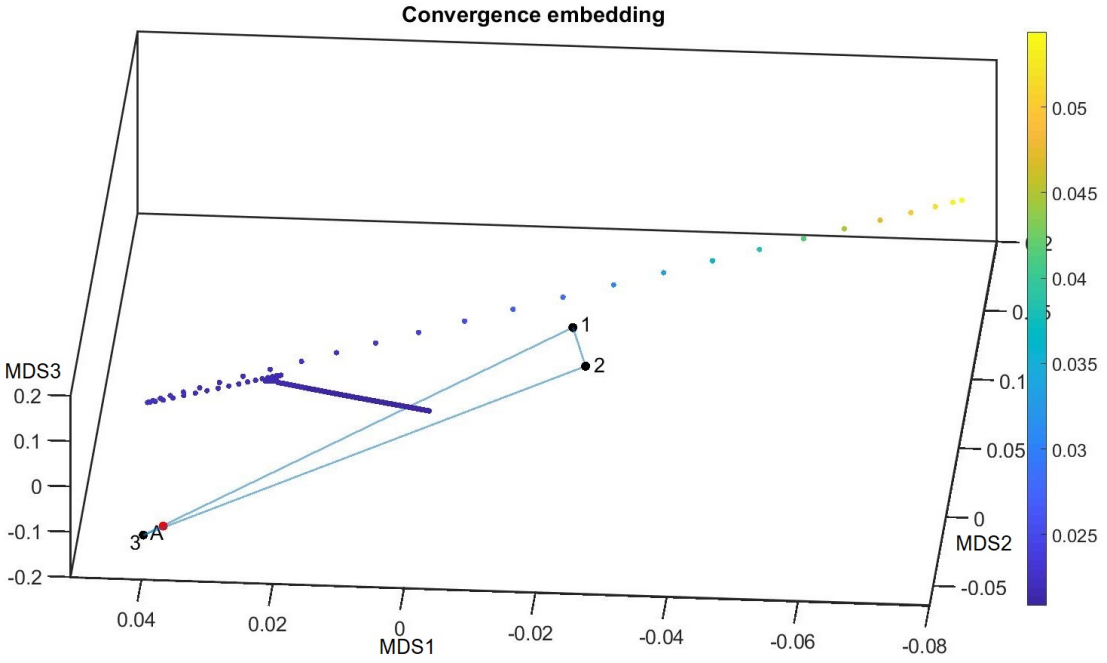

FIGURE 5.27: A visualisation of the networks embedded in $\mathbb{R}^3$ using MDS and the $\sqrt{JS_{\mathcal{N}}}$ as distances. The points labeled 1, 2 and 3 are the input networks shown in Figure 5.25; the point labeled 'A' is the average of the input networks; the remaining points are the complete history of theta coloured with respect to the cost at that time.

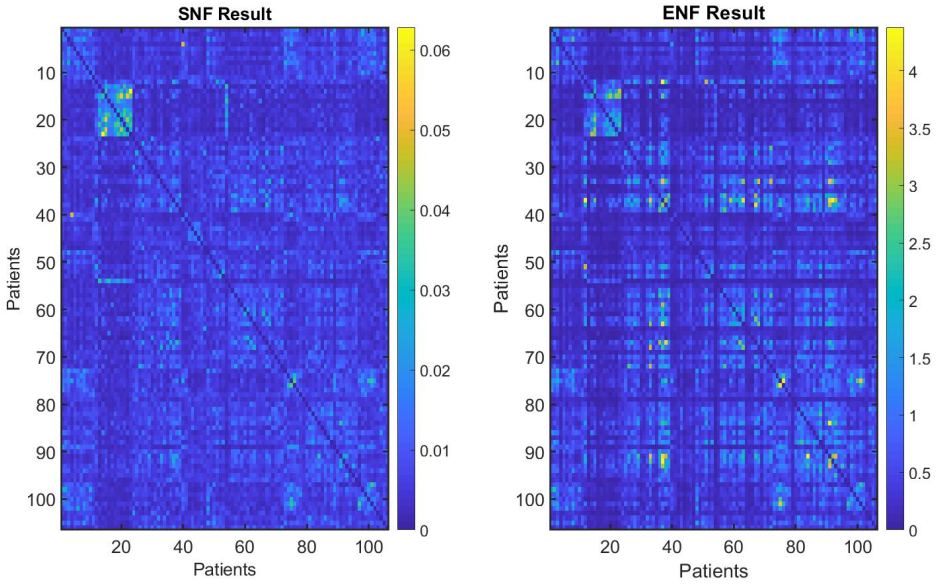

FIGURE 5.28: Heatmaps of the resulting network adjacency matrices obtained from SNF and ENF. Note that the diagonal was removed from both of these, as explained in the Breast cancer section, to improve the resolution for the SNF solution.

The survival profiles for each of the clusters are shown in Figure 5.30 and its survival percentiles in Table 5.37. Blue is clearly the stronger profile, with its 25th percentile being larger than the 75th of all other groups. The remaining three groups have a similar start until approximately month 25 where they being to diverge, with Green becoming the weaker profile, compared to Red, which is reflected in the 75th percentile. We ran the LRT on the clustered results from SNF and ENF and the resulting p-values shown in Table 5.38, both the ENF and SNF are below the 0.05 significance threshold. This means these survival profiles have not been attained by chance and are capturing some biological mechanics at work, showing the effectiveness of ENF. In fact, we also put the other clustering outputs of ENF from the range $[2, \ldots, 6]$ into the LRT and all the p-values obtained, except for $k = 2$, were below the 0.05 significance threshold, shown in Table 5.39.

$\sqrt{JS_{\mathcal{N}}}$	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer1	0	-	-	-	-	-	
Layer2	0.2002	0	-	-	-	-	
Layer3	0.2377	0.292	0	-	-	-	
SNF	0.1164	0.199	0.241	0	-	0.1854	
ENF	0.1090	0.1499	0.1678	0.1295	0	0.1422	↓ 23.3%

TABLE 5.33: Shows the distance, $\sqrt{JS_{\mathcal{N}}}$, between each of the networks and the result from SNF and ENF. The last column denotes the average distance from the solutions to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

CI	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	1	-	-	-	-	-	
Layer 2	0.2133	1	-	-	-	-	
Layer 3	0.0902	0.2142	1	-	-	-	
SNF	0.2324	0.5643	0.3783	1	-	0.3917	
ENF	0.3828	0.5397	0.4001	0.66	1	0.4408	↑ 12.5%

TABLE 5.34: Each of the networks, including the outputs, were clustered into four clusters and then the CI (agreement) between each of the clusterings were calculated. The last column denotes the average concordance between the solutions and the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

NVI	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	0	-	-	-	-	-	
Layer 2	0.8806	0	-	-	-	-	
Layer 3	0.9528	0.88	0	-	-	-	
SNF	0.869	0.608	0.7673	0	-	0.7481	
ENF	0.7636	0.6315	0.7504	0.5131	0	0.7152	↓ 4.4%

TABLE 5.35: Each of the networks, including the outputs, were clustered into four clusters and the distance between each of the clusterings using the NVI metric was calculated. The last column is the average of the distance from the solution to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

NID	Layer 1	Layer 2	Layer 3	SNF	ENF	Average	
Layer 1	0	-	-	-	-	-	
Layer 2	0.7902	0	-	-	-	-	
Layer 3	0.9108	0.787	0	-	-	-	
SNF	0.7851	0.4695	0.6462	0	-	0.6336	
ENF	0.6346	0.4931	0.6221	0.4171	0	0.5833	↓ 7.9%

TABLE 5.36: Each of the networks, including the outputs, were clustered into four clusters and the distance between each of the clusterings using the NID metric was calculated. The last column is the average of the distance from the solution to the input layers. Finally the number to the right shows the percentage decrease/increase of the average ENF value compared to the average SNF value.

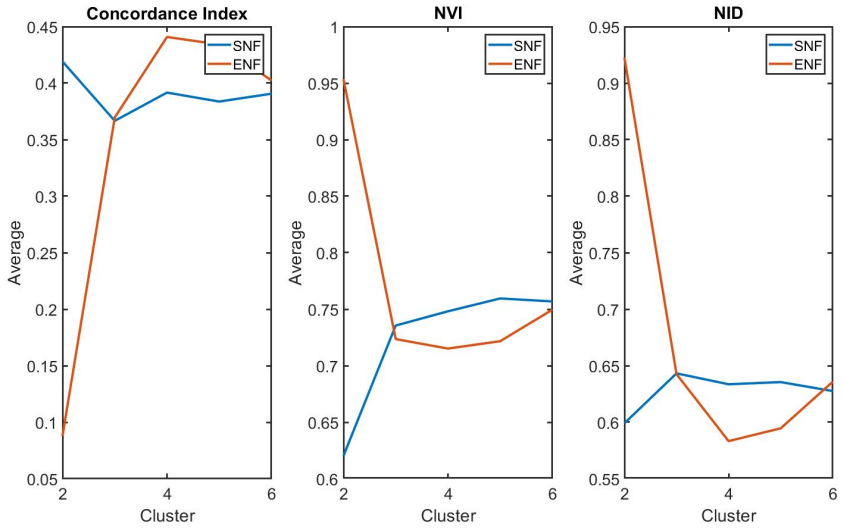

FIGURE 5.29: Tables 5.33 - 5.36 showed the values for CI, NVI and NID for the Lung Cancer dataset for a fixed number of clusters, namely four. Here we show the average values for these quantities for a range of clusters, two to six.

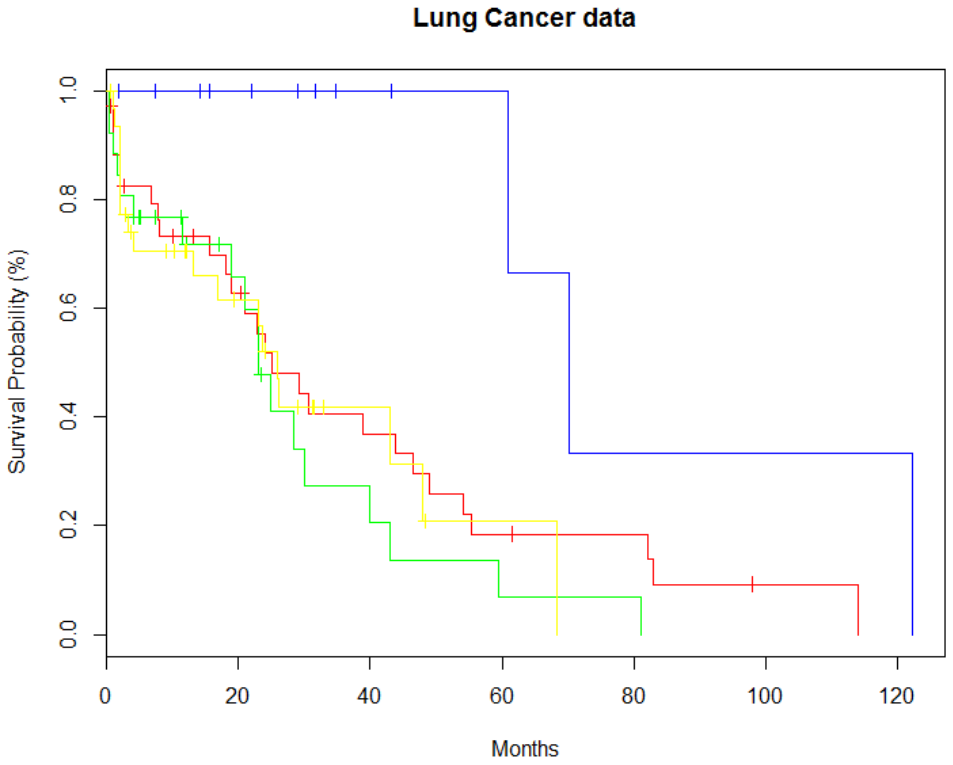

FIGURE 5.30: Kaplan-Meier survival curves for each of the four groups identified from the results of ENF applied to the Lung cancer data. Contains 4 cluster: Red - 36 patients; Blue - 12 patients; Green - 26 patients; Yellow - 32 patients.

Group	$\hat{t}_{25}$	$\hat{t}_{50}$	$\hat{t}_{75}$
Red	8.1	25	54.1
Blue	60.8	70.1	122.3
Green	11.6	23	40
Yellow	3.2	26	47.8

TABLE 5.37: This table shows the respective survival quartiles from the survival curves shown in Figure 5.30.

Method	p-Value
SNF	4.49×10^{-3}
ENF	1.14×10^{-2}

TABLE 5.38: Given the clustering result, (4 clusters), from both ENF and SNF on the Breast data, this table shows the p-values from the LRT.

Clusters	p-value
2	0.2270
3	9.2×10^{-3}
5	2.1×10^{-2}
6	2.1×10^{-2}

TABLE 5.39: This tables shows the p-values from the LRT when the ENF solution is clustered into a different number of clusters.

Chapter 6

aENF

One feature that limits the applicability of ENF is its computational time complexity, since ENF depends on the full computation of all graph Laplacian eigenvalues and eigenvectors. The computation time of which is approximately cubic on the number of vertices, for larger networks, this may become an issue. As we will see in Figure 6.1, computing the precise JSD takes ~ 0.1 seconds for graphs with 200 vertices but rapidly increases to ~ 3 seconds for graphs with 400 vertices. Here, we explain how to approximate network entropy, which originates from the supplementary information of [13], and develop a more computationally efficient method that we call aENF (approximate ENF), specifically designed for large networks. We then show, numerically, how the approximation is significantly faster than the exact version. Then, we calculate the formula for its first derivative and proceed to run aENF, using momentum gradient descent to minimise the function, on two of the cancer data sets, Lung and Glio, to illustrate the benefits to be had from using the approximation. We compare the clustering solutions of ENF and aENF, using our validation metrics, to asses how the quality of the solution changes when we use the approximation. These show that the results from aENF are very similar to those given by ENF, (for one data set they achieve the same clustering output), but overall possess a small and expected decrease in quality. However there is a significant speed increase, 54% and 73% respectively, making aENF more acceptable in its range of applications. (We recommend when the number of nodes are: greater than 300 to use aENF; less than 150 to use ENF and either method for anything in between.)

6.1 Approximating Entropy

Motivated by the definition of Entropy (Eq.3.5), we consider the function $f(\lambda) = \lambda \log_2(\lambda)$, on the range of $[0, 1]$, where $f(0) = 0$, as before. Let us approximate this function by a polynominal of order q:

$$f(\lambda) \approx \sum_{n=0}^{q} \alpha_n \lambda^n. \tag{6.1}$$

where α_n are the coefficients for the terms of the polynomial. These coefficients can be found from a fitting procedure like linear regression that minimises squared errors. The order of the polynomial, q, is a free parameter that the user can set (in our computations, we used q=10). Note that the quality of the fit improves as q grows, at the cost of increased computational time. Inserting this into the definition of entropy for networks (Eq.3.5) gives an approximate entropy (which depends on q):

$$\begin{aligned}
\widetilde{H_{\mathcal{N}}}(G) &= -\sum_{\lambda_i \in \sigma(\bar{L}(G))} \sum_{n=0}^{q} \alpha_n \lambda_i^n \\
&= -\sum_{n=0}^{q} \alpha_n Tr(\bar{L}^n(G)) \\
&= -\alpha_0 N - \sum_{n=1}^{q} \alpha_n Tr(\bar{L}^n(G)).
\end{aligned} \tag{6.2}$$

Recall here that $\bar{L}^n(G)$ is the nth power of the rescaled laplacian matrix of the graph/adjacency matrix G, N is the number of vertices and Tr is the trace of the matrix. Here we have used that if λ is an eigenvalue of $\bar{L}(G)$ then λ^n is a eigenvalue of $\bar{L}^n(G)$ [73] and that the trace is linear. Rather than approximating $f(\lambda)$ for the whole range $[0, 1]$, greater accuracy can be found by using the range $[0, \lambda_{max}]$, where λ_{max} is the largest eigenvalue. This largest eigenvalue can be approximated by $\lambda_{max} < 2d_{max}$, [13, 71, 73] where d_{max} is the largest element along its diagonal, that is, the maximal vertex degree on the network. In [13] they report that for a network with 5000 nodes this yields a speed up by over a factor of 10 and by using $q = 10$ with the restricted range above, produces an error less than 0.1%.

Using this approximation method we can extend it to an approximation of the Jensen-Shannon distance:

$$\widetilde{JS_{\mathcal{N}}}(\theta, G) = \widetilde{H_{\mathcal{N}}}(g\theta + tG) - \frac{\widetilde{H_{\mathcal{N}}}(\theta) + \widetilde{H_{\mathcal{N}}}(G)}{2} \tag{6.3}$$

This gives us the aENF cost function:

$$\begin{aligned}
\widetilde{C}(\{S^{(l)}\}, \theta) &= \frac{1}{l} \sum_{m=1}^{l} \sqrt{\widetilde{JS_{\mathcal{N}}}(S^{(m)}, \theta)}^2 \\
&= \frac{1}{l} \sum_{m=1}^{l} \left[\widetilde{H_{\mathcal{N}}} \left(tS^{(m)} + s^{(m)}\theta \right) \right] - \underbrace{\frac{1}{2l} \sum_{m=1}^{l} [\widetilde{H_{\mathcal{N}}}(S^{(m)})]}_{constant} - \frac{1}{2} \widetilde{H_{\mathcal{N}}}(\theta)
\end{aligned} \tag{6.4}$$

Similar to our previous results, in the next section we calculate the gradients of both parts so the same minimisation method as before can be implemented (gradient descent with momentum). But first we report the computational time speed-up, in Figure 6.1. We generated two Erdős-Rényi random networks, whose edges were connected with probability $p \in [0.1, 0.5]$ and recorded the time taken for computing both the precise distance and the approximation. This was then repeated 100 times and the mean time was then plotted with error bars equal to one standard deviation. Here the effect of computing the full spectrum of eigenvalues becomes apparent and how ENF can become impractical to use for larger networks. Meanwhile by using the approximation a remarkable speed-up is attained, making the complexity looking linear in this range. To see how the computation time of the approximate $JS_{\mathcal{N}}$ increases beyond this range, we performed the same experiment for a selection of larger networks. For this we limited ourselves to 10 repetitions and the mean time and standard deviation is shown in Table 6.1. When plotted and examined alongside Table 6.1 we can conclude this it is roughly cubic, when the nodes are doubled we see an increase by a factor of 8 in the time taken, implying the cubic nature. But as shown in Figure 6.1 this is still significantly faster than the precise $JS_{\mathcal{N}}$. For example the time taken to compute the precise $JS_{\mathcal{N}}$ on a network with 800 nodes (∼26s, shown in Figure 6.1) is roughly the same as the approximate $JS_{\mathcal{N}}$ on 2000 nodes (∼28s shown in Table 6.1), further highlighting the speed up. Therefore by using this approximation it will vastly increase the range of applications that our method can be applied to. Looking at the subplot in Figure 6.1, we would recommend that for applications where the number of nodes are: greater than 300 to use aENF; less than 150 to use ENF and for anything in between left to the user.

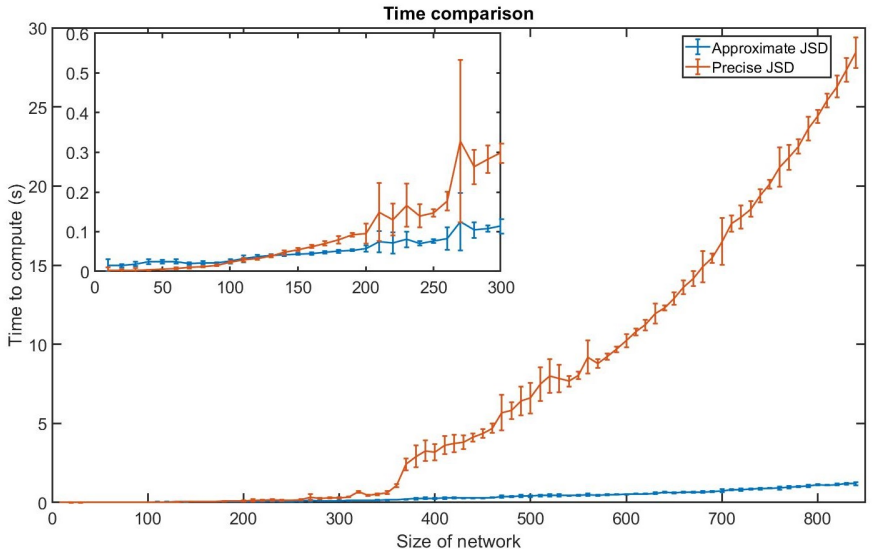

FIGURE 6.1: Computational time for exact and approximate Jensen-Shannon Distance on two random Erdős-Rényi networks. For each number of nodes we took 100 samples and plotted the mean time with error bars equal to one standard deviation. The insert is a blow-up for the results up to 300 nodes.

Nodes	Mean time (s)	Std (s)
1000	4.1	0.2
2000	28	0.9
3000	88.6	1.7
4000	206.3	4.3
5000	398.8	9.5
6000	682.8	14
7000	1077.3	21.9
8000	1606.1	34
9000	2272.3	38

TABLE 6.1: Average computation time for calculating the approximate JSD between larger random networks, using 10 repetitions.

We also considered the error that occurs by using the approximation, specifically for our choice $q = 10$. For this we generated pairs of Erdős-Rényi random networks, with edge probabilities in the range of $p \in [0.10.5]$, and calculated the percentage error in the approximate $JS_{\mathcal{N}}$ compared to the precise $JS_{\mathcal{N}}$. The mean error, with error bars equal to one standard deviation, is shown in Figure 6.2. Here we see the error is in fact fairly small: for the size of networks that we analyses in this thesis we see approximately $1\% - 1.5\%$ error. It also suggests that the growth in the error is linear with respect to the number of nodes in the network along with the size of the error bars. This is to be expected because as networks grow larger there is more room for more complex structures to arise. However, note that this is only for one choice of polynomial degree q and increasing it would further reduce the error in approximating the Jensen-Shannon distance, by increasing the computational time.

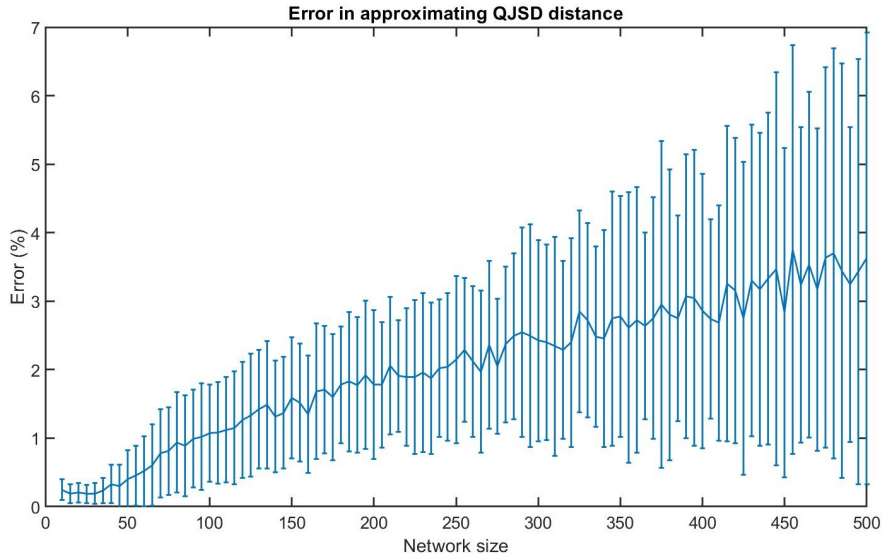

FIGURE 6.2: Using 10 as the degree of the polynomial in the approximation, this figure shows the resulting percentage error when approximating $\sqrt{JS_{\mathcal{N}}}$. For each number of nodes we generate a pair of random Erdős-Rényi network and plotted the mean error with the error bars equal to one standard deviation.

To illustrate this we ran the following numerical experiment. For a selection of polynomial orders q we generated pairs of Erdős-Rényi random networks, on 500 vertices with edge probabilities in the range of $p \in [0.1, 0.5]$, and timed how long it took to compute the approximate $JS_{\mathcal{N}}$. This was repeated 100 times for each q, the final mean time and standard deviation is given in Table 6.2. From it we can see the the computational cost grows approximately linearly.

q	Mean time (s)	Std (s)
5	0.2385	0.0095
10	0.5262	0.0628
15	0.7721	0.0177
20	1.0353	0.0186

TABLE 6.2: The average computation time for calculating the approximate JSD as the order of the polynomial used in the approximation is changed. For each order we generate 100 pairs of Erdős-Rényi networks on 500 vertices.

6.2 Gradient of aENF

In this subsection we now calculate the first derivative of aENF so it can then be minimised by using gradient descent with momentum. As before we split this into the calculation of $\frac{\partial \widetilde{H_{\mathcal{N}}}(\theta)}{\partial \theta_{ij}}$ followed by the calculation of $\frac{\partial \widetilde{H_{\mathcal{N}}}(g\theta+tG)}{\partial \theta_{ij}}$.

6.2.1 Part 1

First we re-write Eq 6.2 in terms of the laplacian, as this were the changes will occur. To do this we start by separating the division by the trace of $L(\theta)$, T:

$$\begin{aligned}\widetilde{H_{\mathcal{N}}}(\theta) &= -\alpha_0 N - \sum_{n=1}^{q} \alpha_n Tr\left(\left(\frac{L(\theta)}{T}\right)^n\right) \\ &= -\alpha_0 N - \sum_{n=1}^{q} \frac{\alpha_n}{T^n} Tr\left(L^n(\theta)\right) \end{aligned} \tag{6.5}$$

Now we differentiate this with respect to the (i,j)-th (and (j,i)-th) entry of θ and from Section 4.4.1 we know the corresponding change in the laplacian is $E_{\mathcal{L}}(i,j)$. Note that the $-\alpha_0 N$ term vanishes as it is a constant and remembering T is also a function of the entries of θ which also needs to be differentiated.

$$\begin{aligned}\frac{\partial\widetilde{H_{\mathcal{N}}}(\theta)}{\partial\theta_{ij}} &= -\sum_{n=1}^{q}\alpha_n\left[\frac{1}{T^n}Tr\left(nL^{n-1}(\theta)E_{\mathcal{L}}(i,j)\right)-\frac{2n}{T^{n+1}}Tr\left(L^n(\theta)\right)\right]\\ &= -\sum_{n=1}^{q}\alpha_n\left[\frac{n}{T}Tr\left(\bar{L}^{n-1}(\theta)E_{\mathcal{L}}(i,j)\right)-\frac{2n}{T}Tr\left(\bar{L}^n(\theta)\right)\right]\\ &= -\frac{1}{T}\sum_{n=1}^{q}n\alpha_n\left[\left(\bar{L}^{n-1}(\theta)_{ii}+\bar{L}^{n-1}(\theta)_{jj}-2\bar{L}^{n-1}(\theta)_{ij}\right)-2Tr\left(\bar{L}^n(\theta)\right)\right] \quad (6.6)\end{aligned}$$

The last step comes from considering only the diagonal entries of the product $\bar{L}^{n-1}(\theta)E_{\mathcal{L}}(i,j)$.

6.2.2 Part 2

Here we calculate the derivative of $\frac{\partial\widetilde{H_{\mathcal{N}}}(g\theta+tG)}{\partial\theta_{ij}}$, starting with as same rearrangement as in Equ 6.5. Proceeding with the differentiation recall from Section 4.4.1 that the induced change in laplacian for this case is $gE_{\mathcal{L}}(k,f)+2L(G)$.

$$\begin{aligned}\frac{\partial\widetilde{H_{\mathcal{N}}}(g\theta+tG)}{\partial\theta_{ij}} &= -\sum_{n=1}^{q}\alpha_n\Big[\frac{1}{T^n}Tr\left(nL^{n-1}(g\theta+tG)(gE_{\mathcal{L}}(k,f)+2L(G))\right)\\ &\qquad -\frac{4gn}{T^{n+1}}Tr\left(L^n(g\theta+tG)\right)\Big]\\ &= -\frac{1}{T}\sum_{n=1}^{q}n\alpha_n\Big[Tr\left(\bar{L}^{n-1}(g\theta+tG)(gE_{\mathcal{L}}(k,f)+2L(G))\right)\\ &\qquad -4gTr\left(\bar{L}^n(g\theta+tG)\right)\Big]\\ &= -\frac{1}{T}\sum_{n=1}^{q}n\alpha_n\Big[g\big(\bar{L}^{n-1}(g\theta+tG)_{ii}+\bar{L}^{n-1}(g\theta+tG)_{jj}\\ &\qquad -2\bar{L}^{n-1}(g\theta+tG)_{ij}\big)+2Tr\left(\bar{L}^{n-1}(g\theta+tG)L(G)\right)\\ &\qquad -4gTr\left(\bar{L}^n(g\theta+tG)\right)\Big] \quad (6.7)\end{aligned}$$

Together this gives us the gradient of aENF

$$\frac{\partial\widetilde{C}(\{G^{(l)}\},\theta)}{\partial\theta_{ij}}=\frac{1}{l}\sum_{m=1}^{l}\left[\frac{\partial\widetilde{H_{\mathcal{N}}}\left(tG^{(m)}+g^{(l)}\theta\right)}{\partial\theta_{ij}}\right]-\frac{1}{2}\frac{\partial\widetilde{H_{\mathcal{N}}}(\theta)}{\partial\theta_{ij}} \quad (6.8)$$

6.3 Precise and Approximate Comparison

To assess the quality of aENF, we applied it to two of the SNF datasets, the Lung and the Glio set. This would also allow for us to compare the precise and approximate solutions and see what losses would occur from using the approximation.

6.3.1 Lung

To make the comparison fair, we gave aENF: the same input layers; initalised θ at the same starting point as used in ENF; the same number of iterations, 7500; used the same value for α, 106, and the same value for the momentum decay $\gamma = 0.9$. Figure 6.3 shows the cost and precise distance to the output from ENF at each iteration of the aENF process, in blue and orange respectively. The cost initalised at 0.0542 and finished at 0.0209 yielding a reduction of 61.4% with a δ-change of 1.6×10^{-12}, this would suggest that the method has successfully converged, which is also supported by the distance to the ENF solution stabilising. To compare the final outputs, Table 6.3 has the cost for both methods, aENF and ENF, for both solutions. We see that aENF produces

	aENF Sol.	ENF Sol.
aENF cost	0.0209	0.0210
ENF cost	0.0209	0.0208

TABLE 6.3: This table shows the output from both aENF and ENF and their respective costs when put into each of the cost functions.

an output that is consistent in cost across both solutions and is also very close in cost to the output of the exact method. This indicates that the two methods have found very similar solutions. This is further supported by the distance to the output of ENF stabilizing at 0.0103. As for the computational time, originally ENF took 6 minutes and 35 seconds to complete for this dataset. For aENF, it took 2 minutes and 54 seconds to complete, yeilding a time reduction of 54%. Together this shows the effectiveness of using the approximation, by obtaining an output that is very close to the precise solution in a significantly smaller amount of time.

Next we evaluated the aENF solution using the validation quantities discussed in Section 5.1. When required, the number of clusters used was 4, to match the analysis in Section 5.2.2. Note that, using eigengaps, the number of clusters would be 3 and then 4 in that order. A concise summary of the results are given in Table 6.4. As it can be seen in the $\sqrt{JS_{\mathcal{N}}}$ values, the approximate solution are very similar to that of the precise solution which is most likely a result of the two solution being so close to each other. In fact, they are so close, that when clustered into 4 clusters they yield *the same* clustering solution. In particular, it has the same values for the CI, NVI and NID.

Furthermore, when these quantities are extended to consider other k in the range $[2, \ldots, 6]$, when $k = 3$ it also obtains *the same* clustering. As we did for ENF, we plot the average for our validation quantities over this cluster range and show the results in Figure 6.4. When $k = 2$ we see that the approximate solution is better than the precise one, however for $k = 5$ and 6 this is not the case. Applying the LRT (log rank test) with the survival data to these clustering, we obtain the p-values in Table 6.5. *All* of these are below the 0.05 significance threshold whereas with the precise method $k = 2$

did not yield a significant p-value. This could be a coincidence or could be a result of the information that is lost by using the approximation. Note also that the p-value for $k = 3$ and 4 are the same as those obtained by the exact solution as they came form the same clustering solution.

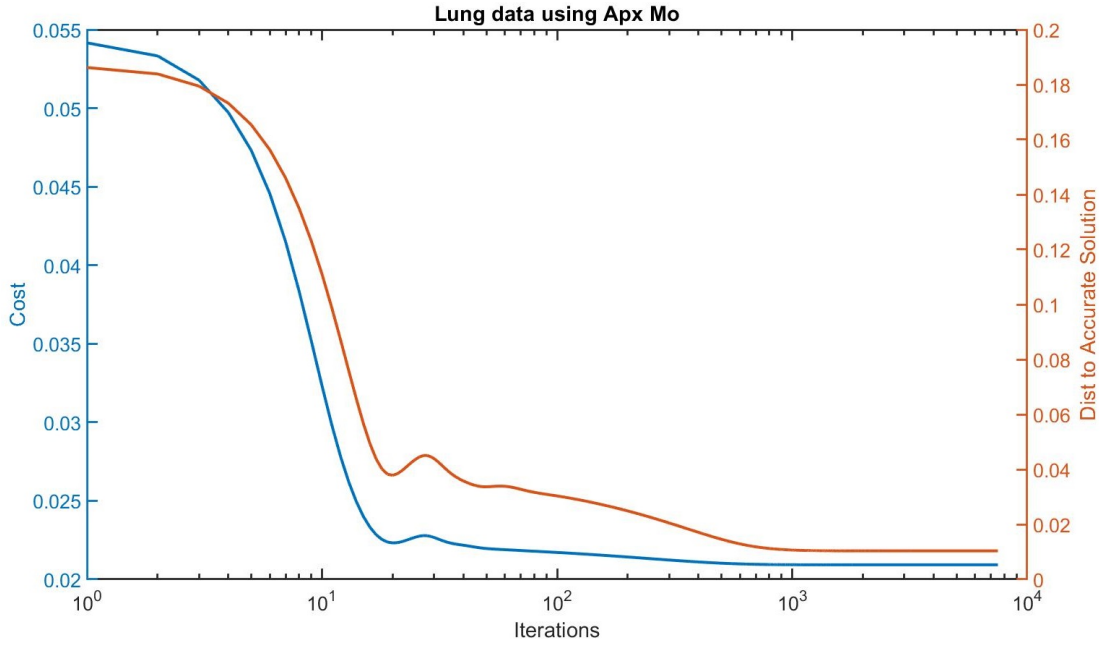

FIGURE 6.3: The blue line shows the cost history of aENF at each iteration when applied to the Lung dataset. The orange line shows the precise distance, $\sqrt{JS_{\mathcal{N}}}$, at each iteration of aENF to output from ENF. Both are given on a log scale.

	Method	Layer 1	Layer 2	Layer 3	ENF	Avg
$\sqrt{JS_{\mathcal{N}}}$	ENF	0.109	0.1499	0.1678	-	0.1422
	aENF	0.1101	0.1499	0.1680	0.0103	0.1427
CI	ENF	0.3828	0.5397	0.4001	-	0.4408
	aENF	0.3828	0.5397	0.4001	1	0.4408
NVI	ENF	0.7636	0.6315	0.7504	-	0.7152
	aENF	0.7636	0.6315	0.7504	0	0.7152
NID	ENF	0.6346	0.4931	0.6221	-	0.5833
	aENF	0.6346	0.4931	0.6221	0	0.5833

TABLE 6.4: Validation quantities for ENF and aENF on the Lung SNF data using 4 clusters

Clusters	p-value
2	0.0022
3	0.0092
4	0.0114
5	0.0165
6	0.0258

TABLE 6.5: p-values for aENF on Lung data

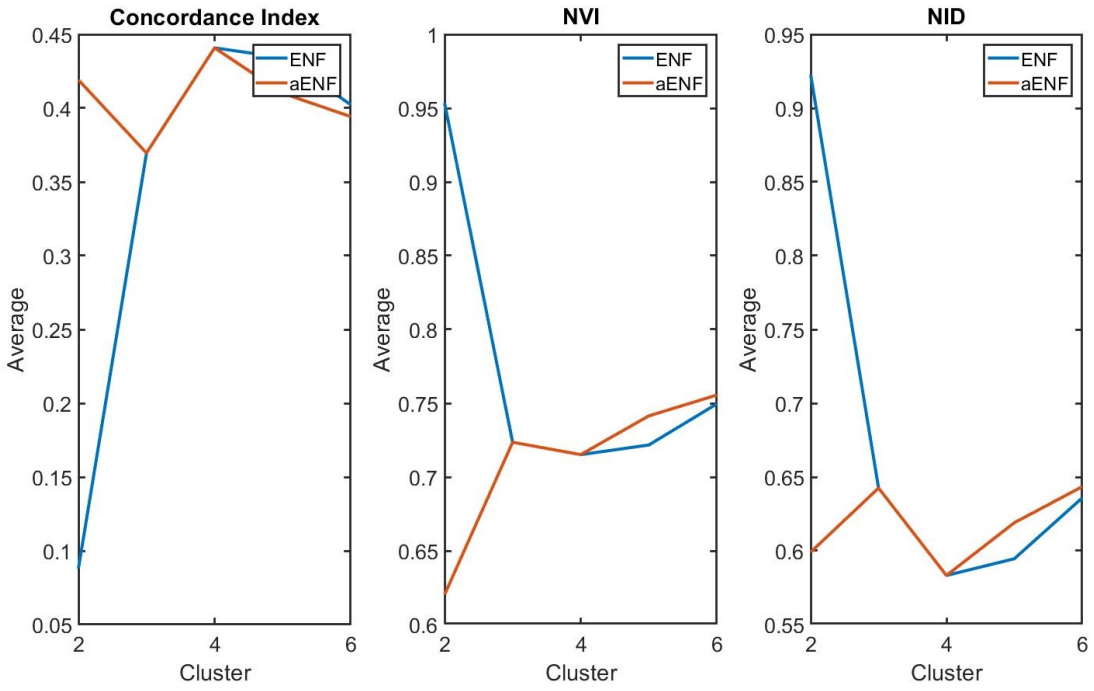

FIGURE 6.4: Tables 6.4 showed the values for CI, NVI and NID for a fixed number of clusters, four. Here we show the average values for these quantities for a range of clusters, two to six.

6.3.2 Glio

For our second dataset, we run aENF on the Glio data set as this was the largest in terms of patients. Again, to make the comparison fair, we gave aENF: the same input layers; initalised θ at the same starting point as used in ENF; the same number of iterations, 15000; used the same value for α, 215, and the same value for the momentum decay $\gamma = 0.9$. Figure 6.5 shows the cost and precise distance to the output from ENF at each iteration of the aENF process, in blue and orange respectively. The cost initalised at 0.0524 and finished at 0.0277 yielding a reduction of 47% with a δ-change of 2.9×10^{-11}, this would suggest that the method has successfully converged, which is also supported by the distance to the ENF solution stabilising. Again, we compare the costs of both solutions with respect to both the aENF and ENF cost function. The results are shown in Table 6.6. We see both solutions are very consistent in the value they obtain and

	aENF Sol.	ENF Sol.
aENF cost	0.0277	0.0278
ENF cost	0.0279	0.0279

TABLE 6.6: This table shows the output from both aENF and ENF and their respective costs when put into each of the cost functions.

actually achieve the same value in the exact ENF. This and the fact that the distance between the aENF and ENF solution is 0.0092, indicates that the two solutions are extremely similar, though not identical. As for the computational time, ENF originally took 1 hour 43 minutes and 37 seconds ($\approx$ 103.5 minutes). For aENF, it took 27 minutes and 58 seconds to complete, yielding a time reduction of 73%. This further highlights the computational advantage of using the approximation.

Again, we next evaluated the aENF solution using the validation quantities, using, when required, $k = 3$ for the number of clusters, as before. If the eigengap were to be used to select the number of clusters then is would be 2 and then 4 in that order. A concise summary of the results are given in Table 6.7. The $\sqrt{JS_{\mathcal{N}}}$ show similar values between the two methods and that the approximate solution is very close to the precise solution. However the effect of these small changes scale with the size of the networks, which is why for the other quantities there is a large amount of agreement but not as much as one would expect. Note we do see that the average of these quantities end up very close to those of the precise solution.

Further, we consider our validation quantities over a larger range of clusters, k in the range $[2, \ldots, 6]$, the averages of which we plot in Figure 6.6. Here we see that all of the values across all of the k are very close to those of the precise solution. This is a result of how close the two solutions are. Applying a the LRT with the survival data to these clustering, we obtain the p-values in Table 6.8. All except $k = 2$ are below the 0.05 significance threshold which is the same as the precise method.

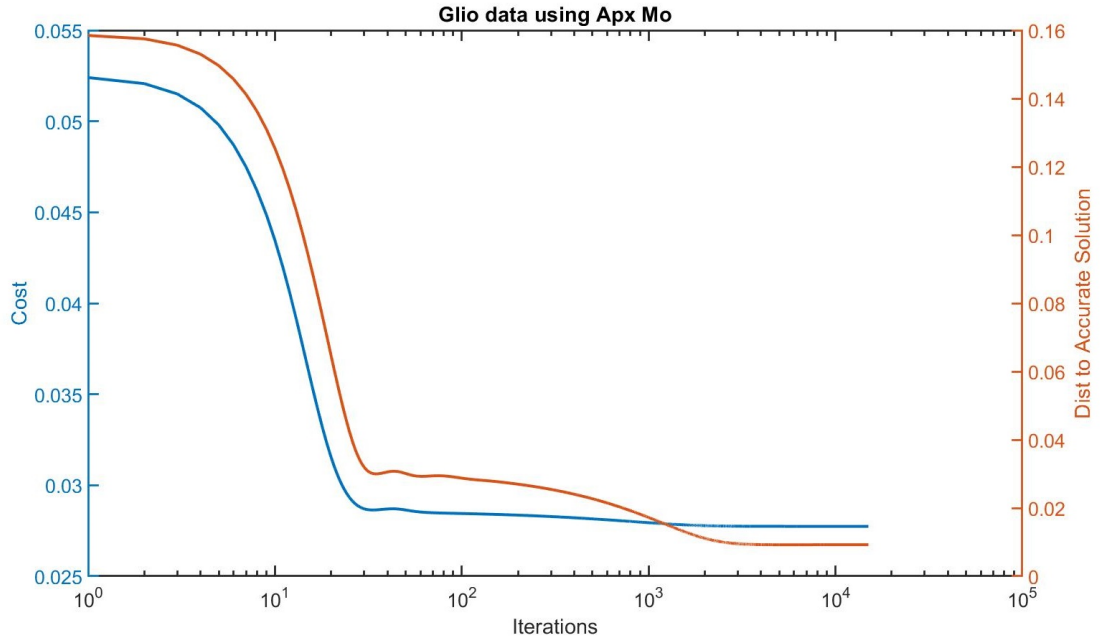

FIGURE 6.5: The blue line shows the cost history of aENF at each iteration when applied to the Glio dataset. The orange line shows the precise distance, $\sqrt{JS_{\mathcal{N}}}$, at each iteration of aENF to output from ENF. Both are given on a log scale.

	Method	Layer 1	Layer 2	Layer 3	ENF	Avg
$\sqrt{JS_{\mathcal{N}}}$	ENF	0.1677	0.1702	0.1627	-	0.1669
	aENF	0.1686	0.1687	0.1641	0.0092	0.1671
CI	ENF	0.1484	0.1327	0.1793	-	0.1535
	aENF	0.1235	0.1527	0.1768	0.8012	0.1510
NVI	ENF	0.9199	0.9325	0.9018	-	0.9181
	aENF	0.9342	0.9211	0.9032	0.3319	0.9195
NID	ENF	0.8558	0.9031	0.933	-	0.864
	aENF	0.878	0.8865	0.8324	0.2127	0.8656

TABLE 6.7: Validation quantities for ENF and aENF on the Glio SNF data using 3 clusters

Clusters	p-value
2	0.361
3	0.0036
4	0.0034
5	0.0117
6	0.0014

TABLE 6.8: p-values for aENF on glio data

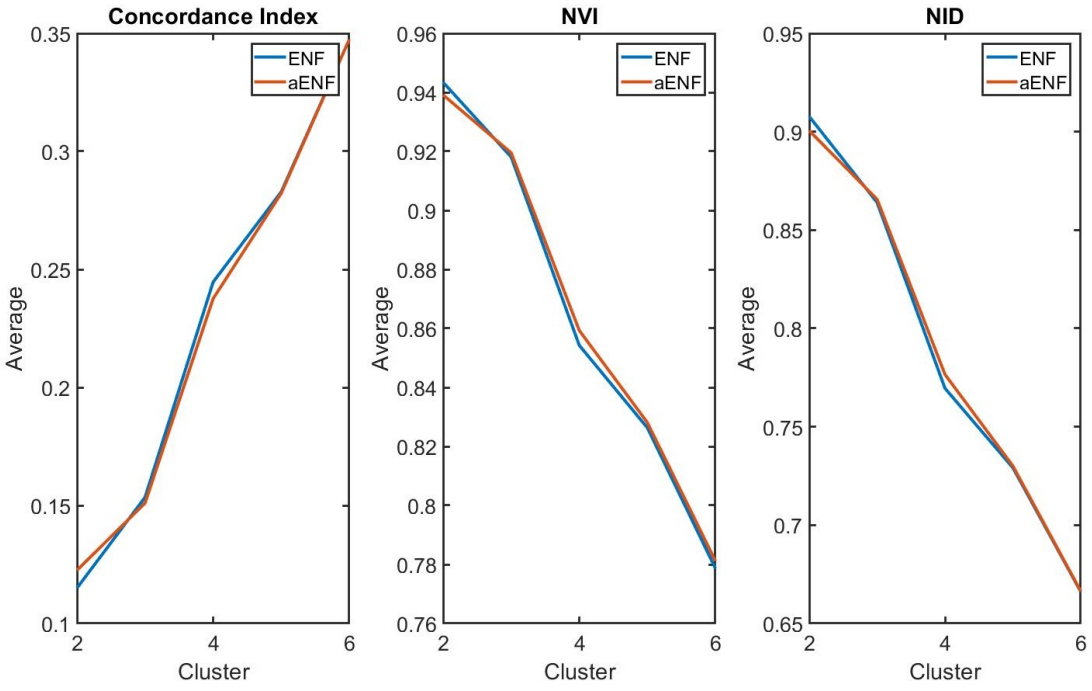

FIGURE 6.6: Tables 6.7 showed the values for CI, NVI and NID for a fixed number of clusters, three. Here we show the average values for these quantities for a range of clusters, two to six.

Chapter 7

Conclusion and Future Work

In summary this project set out to develop a new network based method for the integration of network-based data sets with the objective of clustering the samples the data originate from, motivated by data integration of 'omic patient data sets. We started by investigating an existing state-of-the-art data integration method in bioinformatics, SNF [14]. First, we investigated whether a limit could be taken as originally it is an iterative process. We showed that the iterative process on which SNF is based converges to a constant matrix, hence this method is not robust in the limit.

The main result in this thesis is a new network based integration method, which we called Entropy Network Fusion. It is based on a distance function (Quantum Jensen-Shannon Distance, or QJSD) based on network entropy. This, in turn, uses the eigenvalues of the normalised Laplacian of a network as proxies of their clustering structure of the network. The intuition is that, given the same number of vertices, the closer two networks are with respect to their clustering structure, the smaller their QJSD is. ENF finds the network with a clustering structure (measured by their pairwise QJSD) that is closest to all of the given input networks, with respect to the sum of squared distances. The input of ENF are m arbitrary n by n real symmetric matrices representing m measures of pairwise similarity between n objects. (It is up to the user to construct these similarity matrices.) The cost function (sum of squared QJSDs to input networks) is minimised using gradient descent with momentum, which is controlled by a learning rate α, and a decay term γ to control the momentum. We have also showed that the function is convex, meaning that there is only one optimal solution for a given set of input networks and regardless of the initial starting point it will all converge to that one optimal solution.

When we tested our method, we used the data from the SNF paper [14] and compared our method to SNF. As ENF is flexible with its inputs the input layers that are generated by SNF were duplicated and given to ENF as its inputs, to make a fair comparison between the two methods. We selected three clustering validation metrics (CI, NVI and NID) to compare the clustering structures of the outputs and the inputs, and verify

which method produced an output whose clustering structure was more aligned with the inputs. For these validation metrics overall we saw that the ENF results were better than those of SNF, except for the Glio dataset. This means that ENF finds a solution that is structurally closer to its inputs and capturing more of the information provided in them. The Glio set, whilst is slightly worse for the particular k, has very similar values to that of SNF across the full cluster range we examined. In the survival analysis we saw all of the p-values obtained from the LRT were below the 0.05 significance level, with p-values lower than SNF for 2 of the cancer data sets (Colon and Kidney). Furthermore, for all of the datasets except Kidney, ENF also had other significant p-values in the cluster range we examined. This shows ENF's ability to find significant structure in the data/from the networks that has a real world impact and can be used to make a difference.

The most computationally demanding step in ENF is the calculation of the laplacian eigenvalues, which has in general cubic time complexity on the number of nodes (patients). This is a potential restriction for large networks (typical genetic networks can have over $10,000$ nodes/genes). In order to address this issue, we developed an approximate version of ENF, called aENF, which can run on larger networks in an acceptable time. Using a polynomial approximation we saw an expected decrease in the quality of results, in comparison to the exact method, but aENF still produced significant p-values. This shows that whilst the finer details of the solution maybe lost with the approximation, the overall clustering structure that it finds is still significant. It also achieves massive acceleration in computation time, achieving a reduction of 54% and 73%. For example, on the Glio data, the aENF method could have run for approximately 4 time longer and would have finished at the same time as the ENF method. By which point it may have matched or surpassed the quality of exact method. This trade-off, small decrease in quality for a big speed reduction, is highly worthwhile and opens the door to a larger range of application of ENF and aENF.

Our investigation into the use of network entropy in data integration suggested several lines of research that we did not have time to explore. Here we describe some of those.

- **Barycenter shortcut.** For a collection of points in euclidean space, the point that minimises the squared distance to all of the points is the barycenter, the arithmetic mean of the points. Our cost function also minimises a sum of squared distances but with respect to a different metric, $\sqrt{JS_{\mathcal{N}}}$. If an appropriate analogy of the barycenter with respect to this metric could be found then ENF could utilise this shortcut and be changed from an iterative procedure to a direct one. As we saw in the MDS embedding of the ENF results, all of the solutions converged to the barycenter of the triangle formed by the input layers. This could greatly decrease its computational time and increase the range of applications for the precise variation and also help with the other versions.

- **Matrix cost function.** In the ENF cost function, we explicitly compute the eigenvalues, and this becomes the most costly computational step. However if the matrix formulation, (Eq. 3.3), were used in the cost function instead, this could result in speed-accuracy trade off. The loss of accuracy results from the approximate calculation of $\log(\rho)$, which for a matrix is given by an infinite sum. On the other hand, the computational speed gain comes from avoiding the full eigenvalue computation and taking matrix products instead. This could potentially lead to a version of ENF whose performance, with respect to speed and quality, would be between that of ENF and aENF.

- **Directed networks.** As mentioned in Section 4.2 there has been recent work extending entropy to directed networks [79]. Directed networks can be more informative than undirected networks and appear naturally in applications. For example k-nearest neighbours (KNN), which is a popular classification/regression method, when applied to a network, does not produce in general a symmetric network. (If point x is among the KNN of point y there is no guarantee that the reverse is true.) We would like to extend ENF to accept directed networks as input.

- **Higher dimensions.** Networks are a basic one-dimensional representation of data (representing objects are nodes and pairwise similarities as edges, which are 0- respectively 1-dimensional geometrical or topological objects). These can be further generalised into higher dimensional objects called simplicial complexes [87], and potentially capture more subtle (higher-order) relationships within the data that is be lost in a network representation. For example a 2-dimensional object would consider the similarity of 3 patients at the same time rather than the 3 sets of individual pairwise similarities. This could reveal more subtle relationships, and new/further subtypes of cancer. Similar as to when Shen et al [12] found a new breast cancer subtype by performing DI.

- **Time series data** Another avenue for research would be to investigate the application of our work to time series data. Over time, patient data will change as for example, their illness takes its course. As our method is a general network data integration method, it can be applied to this or any other time series network data. By using our method we could potentially identify groups of objects or patients that are consistently grouped together over the course of time, if they exist, providing new insights.

- **Feature Extraction.** Given the results shown in Section 5.2.2, four out of the five data sets had statistically significant p-values, this could be explored further to find the defining features of the clusters. If, for each cluster, a defining set of features can be found in the original data they can be compared to existing knowledge, potentially providing new insights. This process is called feature extraction [88].

These defining features could then be used to build a ML classifier so when a new patient needs to be clustered (classified), rather requiring all the features from the patient, only the defining features need to be gathered. This would save time, resources and reduced the time to provide the appropriate treatment to the patient.

Appendix A

Laplacian Identities

The following are technical results that are used in the derivation of the gradient for the ENF cost function in Section 4.4.

Given a Graph G, the degree matrix of G, D_G is linear with respect to addition and multiplication by a scalar:

$$\begin{aligned} D_{cG,ii} &= \sum_j cG_{ij} \\ &= c\sum_j G_{ij} \\ &= cD_{G,ii} \end{aligned} \tag{A.1}$$

$$\begin{aligned} D_{G+M,ii} &= \sum_j (G+M)_{ij} \\ &= \sum_j G_{ij} + \sum_j M_{ij} \\ &= D_{G,ii} + D_{M,ii} \end{aligned} \tag{A.2}$$

Consider the laplacian as defined in Definition 3.2. Thanks to what we showed above, the laplacian is also linear with respect to addition and multiplication by scalar. We have:

$$\begin{aligned} L(cG) &= D_{cG} - cG \quad \text{for } c \in \mathbb{R} \\ &= cD_G - cG \\ &= cL(G) \end{aligned} \tag{A.3}$$

$$\begin{aligned} L(G+A) &= D_{G+A} - (G+A) \\ &= D_G + D_A - G - A \\ &= L(G) + L(A) \end{aligned} \tag{A.4}$$

Now the re-scaled laplacian is defined as:

$$\bar{L}(G) = \frac{L(G)}{Tr(L(G))} = \frac{D_G - G}{\sum_{ij} G_{ij}} \tag{A.5}$$

Where $Tr(L(G)) = \sum_{ij} G_{ij} = \sum_{\lambda_k \in \sigma(L(G))} \lambda_k$. Now we will show that the rescaled laplacian *is not* linear with respect to addition and multiplication.

$$\begin{aligned}
\bar{L}(cG) &= \frac{D_{cG} - cG}{\sum_{ij} cG_{ij}} \quad \text{for } c \in \mathbb{R} \\
&= \frac{c(D_G - G)}{c\sum_{ij} G_{ij}} \\
&= \frac{D_G - G}{\sum_{ij} G_{ij}} \\
&= \bar{L}(G)
\end{aligned} \tag{A.6}$$

Here we have shown that multiplication by any scalar results in the same rescaled laplacian.

$$\begin{aligned}
\bar{L}(G+A) &= \frac{D_{G+A} - (G+A)}{\sum_{ij}(G_{ij} + A_{ij})} \\
&= \frac{(D_G - G) + (D_A - A)}{g + a} \quad \text{setting } g = \textstyle\sum_{ij} G_{ij} \text{ and } a = \sum_{ij} A_{ij} \\
&= \bar{L}(G) + \frac{a}{g+a}\left[\bar{L}(A) - \bar{L}(G)\right] && \text{(A.7)} \\
&= \frac{g}{g+a}\bar{L}(G) + \frac{a}{g+a}\bar{L}(A) && \text{(A.8)} \\
&\neq \bar{L}(G) + \bar{L}(A) && \text{(A.9)}
\end{aligned}$$

Hence we see that the rescaled laplacian of a sum of matrices is not equal to the sum of respective laplacian matrices. However we can show that the average of two rescaled laplacians is equal to the laplacian of a weighted sum. Consider the following:

$$\begin{aligned}
\frac{\bar{L}(G) + \bar{L}(A)}{2} &= \frac{1}{2}\left[\frac{D_G - G}{g} + \frac{D_A - A}{a}\right] \\
&= \frac{1}{2}\left[\frac{aD_G - aG + gD_A - gA}{ag}\right] \\
&= \frac{D_{aG+gA} - (aG + gA)}{2ag} \\
&= \bar{L}(aG + gA)
\end{aligned} \tag{A.10}$$

where $g = \sum_{ij} G_{ij}$ and $a = \sum_{ij} A_{ij}$.

Appendix B

Limit of a Matrix

The following are technical results for calculating the limit of a matrix, which is required in Chapter 2.2. The results used in this section come from *"Matrix Analysis"* by Horn and Johnson [71]. Consider the set of all n by m matrices with real entries, which shall be denoted by $M_{n,m}$. If $n = m$ then this is written as M_n.

Definition B.1. Given a matrix $A \in M_n$, a vector $X \in \mathbb{C}^n$ and a scalar $\lambda \in \mathbb{C}$ such that they satisfy:

$$Ax = \lambda x$$

then λ is called an *eigenvalue* of A and x is its associated *eigenvector*. For a given matrix $A \in M_n$ the set of all the eigenvalues of A, are called the *spectrum of* A and is denoted by $\sigma(A)$. The *spectral radius* of a matrix A is defined as $\rho(A) = max\{|\lambda| : \lambda \in \sigma(A)\}$.

Definition B.2. A matrix $A \in M_n$ is said to be *reducible* if either:

1) $n = 1$ and $A = 0$, or

2) $n \geq 2$ and there is a permutation matrix $P \in M_n$ and some integer r with $1 \leq r \leq n - 1$ such that:

$$P^T AP = \begin{bmatrix} B & C \\ 0 & D \end{bmatrix} \tag{B.1}$$

where $B \in M_r, D \in M_{n-r}, C \in M_{r,n-r}$ and $0 \in M_{n-r,r}$ is a zero matrix.

A matrix $A \in M_n$ is said to be *irreducible* if it is not reducible.

Definition B.3. A directed graph Γ is *strongly connected* if between every pair of distinct nodes v_i, v_j in Γ there is a directed path that begins at v_i and end at v_j.

Note that every matrix, $A \in M_n$, can be interpreted as an adjacency matrix of a graph on n vertices. Given an entry, A_{ij}, this is the weight of the edge connecting vertex i to

vertex j. If the entry is equal to zero then there is no edge between the two vertices's. Similarly if a entry $A_{ij} \neq A_{ji}$ then there are directed edges between vertex i and vertex j, or only one edge if either is equal to 0.

Definition B.4. The directed graph of $A \in M_n$ denoted by $\Gamma(A)$ is the directed graph on n nodes $v_1, \ldots, v_n$ such that there is a directed arc in $\Gamma(A)$ from v_i to v_j with weight a_{ij} if and only if $a_{ij} \neq 0$.

Definition B.5. A matrix is said to be *strongly connected* if and only if the directed graph $\Gamma(A)$ is strongly connected.

Definition B.6. A matrix $A \in M_n$ is said to be *nonnegative*, written $A \geq 0$, if all of its entries a_{ij} are nonnegative. Similarly we say a matrix $A \in M_n$ is *positive*, $A > 0$, if all of its entries a_{ij} are positive.

Definition B.7. For any given $A = [a_{ij}] \in M_{m,n}$ we define $abs(A) = [|a_{ij}|]$ and $I(A) = [\mu_{ij}]$ in which $\mu_{ij} = 1$ if $a_{ij} \neq 0$ and $\mu_{ij} = 0$ if $a_{ij} = 0$. The matrix $I(A)$ is called the *indicator matrix* of A.

Theorem B.8. *Let $A \in M_n$. The following are equivalent:*

1) A is irreducible.

2) $(I + abs(A))^{n-1} > 0$.

3) $[I + I(A)]^{n-1} > 0$.

4) $\Gamma(A)$ is strongly connected.

5) A is strongly connected.

Proof. For a proof of this theorem the reader is referred to Theorem 6.2.24 in [71]. □

Theorem B.9. *Given an $A \in M_n$ that is irreducible and nonnegative then:*

1) $\rho(A) > 0$.

2) $\rho(A)$ is an eigenvalue of A.

3) There is a positive vector x such that $Ax = \rho(A)x$.

4) $\rho(A)$ is a algebraically (and geometrically) simple eigenvalue of A.

Proof. For a proof of this lemma the reader is referred to Theorem 8.4.4 in [71]. □

Definition B.10. The vector x that satisfies $Ax = \rho(A)x$, whose components sum to 1, is called the *(right) Perron vector*. There is analogous definition for the *(left) Perron vector*, the vector y that satisfies $yA = \rho(A)y$, whose components sum to 1.

Definition B.11. A nonnegative matrix $A \in M_n$ is called *primitive* if it is irreducible and the multiplicity of the eigenvalue with largest modulus is one.

Theorem B.12. *Given a matrix $A \in M_n$ that is nonnegative and primitive, there is x and y such that $Ax = \rho(A)x, A^T y = \rho(A)y, x > 0, y > 0$ and $x^T y = 1$. Then*

$$\lim_{m\to\infty} [\rho(A)^{-1}A]^m = L > 0 \tag{B.2}$$

where $L = xy^T$.

Proof. For a proof of this theorem the reader is referred to Theorem in 8.5.1 [71]. □

Theorem B.13. *If $A \in M_n$ is nonnegative then A is primitive if and only if $A^m > 0$ for some $m \geq 1$.*

Proof. For a proof of this theorem the reader is referred to Theorem 8.5.2 in [71]. □

Lemma B.14. *If $A \in M_n$ is nonnegative and irreducible, and if all the main diagonal entries of A are positive then $A^{n-1} > 0$, so A is primitive.*

Proof. For a proof of this lemma the reader is referred to Lemma 8.5.4 in [71]. □

Appendix C

Second Derivative of ENF

In this appendix we calculate the second derivative of our cost function, namely

$$\frac{\partial C(G^{(l)},\theta)}{\partial\theta_{\alpha\beta}\partial\theta_{ij}} = \frac{1}{l}\sum_{m=1}^{l}\left[\frac{\partial H_{\mathcal{N}}\left(tG^{(m)}+g^{(l)}\theta\right)}{\partial\theta_{\alpha\beta}\partial\theta_{ij}}\right] - \frac{1}{2}\frac{\partial H_{\mathcal{N}}(\theta)}{\partial\theta_{\alpha\beta}\partial\theta_{ij}}$$

Although we do not use this due to its computational complexity we include it for completeness.

C.1 Eigenvectors

When calculating the second derivative of our cost function we need to differentiate eigenvectors with respect to the entry of the matrix. For this we use the results on how to calculate these eigenvector derivatives from [89,90]. Given an eigenvector, x_i, of a real symmetric matrix, whose norm is equal to one, then the eigenvector derivative can be expressed as a linear combination of the eigenvectors. Specifically, let E be a perturbation in our matrix, M and consider $M+tE$ for a real parameter t. Assuming M has no repeated eigenvalues, which is the case in most empirical matrices, then a eigenvector x_i has a derivative of:

$$\frac{\partial x_i}{\partial E} = \sum_{k=1,k\neq i}^{n}\Omega_{ik,E}x_k \tag{C.1}$$

where

$$\Omega_{ik,E} = \frac{x_k^t E x_i}{\lambda_i-\lambda_k}. \tag{C.2}$$

Now we consider the two cases we have in out cost function $H_{\mathcal{N}}(G)$ and $H_{\mathcal{N}}(aG+gA)$. When we have $H_{\mathcal{N}}(G)$ we know the change in the laplaican is $E_{\mathcal{L}}(k,f)$, as we worked

in Subsection 4.4.1. Therefore in this case we have the derivatives being:

$$\frac{\partial x_i}{\partial G_{kf}} = \sum_{k=1,k\neq i}^{n} \frac{x_k^t E_{\mathcal{L}}(k,f) x_i}{\lambda_i - \lambda_k} x_k \tag{C.3}$$

For our other case, $H_{\mathcal{N}}(aG + gA)$, we calculated the resulting change in the matrix to be $aE_{\mathcal{L}}(k,f) + 2L(A)$, also in Subsection 4.4.1. Thus for this other case we have the derivatives being:

$$\frac{\partial x_i}{\partial G_{kf}} = \sum_{k=1,k\neq i}^{n} \frac{x_k^t (aE_{\mathcal{L}}(k,f) + 2L(A)) x_i}{\lambda_i - \lambda_k} x_k \tag{C.4}$$

C.2 Second Derivative

With the derivatives of eigenvectors calculated we now proceed to calculate the second derivative of the ENF cost function.

C.2.1 Part 1: $H_{\mathcal{N}}(G)$

Recall from Eq 4.17 that the first derivative for is $H_{\mathcal{N}}(G)$:

$$\frac{\partial H_{\mathcal{N}}(G)}{\partial G_{ij}} = \frac{-1}{T} \sum_{k=1}^{n} \left[(x_{ik} - x_{jk})^2 \log_2 (\lambda_k) \right] + \frac{2}{T} \log_2(T) - \frac{2}{T} H_{\mathcal{N}}(G)$$

Now we differentiate again to obtain the second derivative. To keep it from becoming unclear, we do this part by part.

$$\begin{aligned}
\frac{\partial}{\partial G_{ab}} \left(\frac{-2}{T} H_{\mathcal{N}}(G) \right) &= \frac{2}{T^2} H_{\mathcal{N}}(G) \sum_{k=1}^{n} \frac{\partial \lambda_k}{\partial G_{ab}} + \frac{-2}{T} \frac{\partial H_{\mathcal{N}}(G)}{\partial G_{ab}} \\
&= \frac{2}{T^2} H_{\mathcal{N}}(G) \sum_{k=1}^{n} (x_{ak} - x_{bk})^2 + \frac{-2}{T} \frac{\partial H_{\mathcal{N}}(G)}{\partial G_{ab}} \\
&= \frac{4}{T^2} H_{\mathcal{N}}(G) + \frac{-2}{T} \frac{\partial H_{\mathcal{N}}(G)}{\partial G_{ab}} \\
&= \frac{-2}{T} \left(\frac{-2}{T} H_{\mathcal{N}}(G) + \frac{\partial H_{\mathcal{N}}(G)}{\partial G_{ab}} \right)
\end{aligned}$$

$$\frac{\partial}{\partial G_{ab}}\left(\frac{2}{T}\log_2(T)\right) = \frac{-2}{T^2}\log_2(T)\sum_{k=1}^{n}\frac{\partial\lambda_k}{\partial G_{ab}} + \frac{2}{T^2\ln(2)}\sum_{k=1}^{n}\frac{\partial\lambda_k}{\partial G_{ab}}$$
$$= \frac{-4}{T^2}\log_2(T) + \frac{4}{T^2\ln(2)}$$
$$= \frac{-2}{T}\left(\frac{2}{T}\log_2(T) - \frac{2}{T\ln(2)}\right)$$

$$\frac{\partial}{\partial G_{ab}}\left(\frac{-1}{T}\sum_{k=1}^{n}\left[(x_{ik}-x_{jk})^2\log_2(\lambda_k)\right]\right) = \frac{1}{T^2}\sum_{k=1}^{n}\left[(x_{ik}-x_{jk})^2\log_2(\lambda_k)\right]\sum_{k=1}^{n}\left[\frac{\partial\lambda_k}{\partial G_{ab}}\right] +$$
$$\frac{-1}{T}\sum_{k=1}^{n}\left[\frac{(x_{ik}-x_{jk})^2}{\lambda_k\ln(2)}\frac{\partial\lambda_k}{\partial G_{ab}} + \right.$$
$$\left. 2(x_{ik}-x_{jk})\log_2(\lambda_k)\left(\frac{\partial x_{ik}}{\partial G_{ab}} - \frac{\partial x_{jk}}{\partial G_{ab}}\right)\right]$$

Combing these three terms and simplifying we get:

$$\frac{\partial H_{\mathcal{N}}(G)}{\partial G_{ab}\partial G_{ij}} = \frac{-2}{T}\left(\frac{\partial H_{\mathcal{N}}(G)}{\partial G_{ab}} + \frac{\partial H_{\mathcal{N}}(G)}{\partial G_{ij}}\right) + \frac{4}{T^2\ln(2)} - \frac{1}{T}\sum_{k=1}^{n}\left[\frac{(x_{ik}-x_{jk})^2(x_{ak}-x_{bk})^2}{\lambda_k\ln(2)}\right.$$
$$\left. + 2(x_{ik}-x_{jk})\log_2(\lambda_k)\left(\frac{\partial x_{ik}}{\partial G_{ab}} - \frac{\partial x_{jk}}{\partial G_{ab}}\right)\right]$$

where $\frac{\partial H_{\mathcal{N}}(G)}{\partial G_{ab}\partial G_{ij}} = 0$ if $i = j$ or $a = b$ (or both) and the eigenvector derivatives are calculated according to Eq C.3.

C.2.2 Part 2: $H_{\mathcal{N}}(aG + gA)$

Recall from Eq 4.18 that the first derivative for $H_{\mathcal{N}}(aG + gA)$ is

$$\frac{\partial H_{\mathcal{N}}(aG+gA)}{\partial G_{ij}} = \frac{-1}{T}\sum_{k=1}^{n}\log_2(\lambda_k)\left[a(x_{ik}-x_{jk})^2 + 2x_k^T L(A)x_k\right]$$
$$+ \frac{2}{T}\left[\log_2(T) - H(aG+gA)\right]\left[a + \sum_{k=1}^{n}x_k^T L(A)x_k\right]$$

Next, like in the first case, we now differentiate the above expression to obtain the second derivative. To keep it clear, we break it apart as follows:

$$\frac{\partial H(aG+gA)}{\partial G_{\alpha\beta}\partial G_{ij}} = \frac{\partial}{\partial G_{\alpha\beta}}\Bigg(\underbrace{\frac{-1}{T}\sum_{k=1}^{n}\log_2(\lambda_k)\left[a(x_{ik}-x_{jk})^2 + 2x_k^T L(A)x_k\right]}_{P_1} +$$
$$\underbrace{\frac{2}{T}\left[\log_2(T) - H(aG+gA)\right]\left[a + \sum_{k=1}^{n}x_k^T L(A)x_k\right]}_{P_2}\Bigg)$$

Now resolving these two parts:

$$P_1 = \frac{2(a+\sum_{k=1}^{n} x_k^T L(A) x_k)}{T^2} \sum_{k=1}^{n} \log_2(\lambda_k) \left[a(x_{ik}-x_{jk})^2 + 2x_k^T L(A) x_k\right] + \\ - \frac{1}{T}\sum_{k=1}^{n}\left[\frac{\left[a(x_{\alpha k}-x_{\beta k})^2 + 2x_k^T L(A)x_k\right]\left[a(x_{ik}-x_{jk})^2 + 2x_k^T L(A)x_k\right]}{\lambda_k \ln(2)}\right. \\ \left. + \log_2(\lambda_k)\left[4x_k^T L(A)\frac{\partial x_k}{\partial G_{\alpha\beta}} + 2a(x_{ik}-x_{jk})(\frac{\partial x_{ik}}{\partial G_{\alpha\beta}} - \frac{\partial x_{jk}}{\partial G_{\alpha\beta}})\right]\right]$$

$$P_2 = -\frac{4(a+\sum_{k=1}^{n} x_k^T L(A) x_k)}{T^2}\left[\log_2(T) - H(aG+gA)\right]\left[a + \sum_{k=1}^{n} x_k^T L(A) x_k\right] \\ + \frac{2}{T}\left[\frac{2(a+\sum_{k=1}^{n} x_k^T L(A) x_k)}{T\ln(2)} - \frac{H(aG+gA)}{\partial G_{\alpha\beta}}\right]\left[a + \sum_{k=1}^{n} x_k^T L(A) x_k\right] \\ + \frac{2}{T}\left[\log_2(T) - H(aG+gA)\right]\left[2\sum_{k=1}^{n} x_k^T L(A)\frac{\partial x_k}{\partial G_{\alpha\beta}}\right]$$

Combing these back together we get

$$\Rightarrow \frac{\partial H(aG+gA)}{\partial G_{\alpha\beta}\partial G_{ij}} = -\frac{2(a+\sum_{k=1}^{n} x_k^T L(A) x_k)}{T^2}\left(\frac{H(aG+gA)}{\partial G_{ij}} + \frac{H(aG+gA)}{\partial G_{\alpha\beta}}\right) + \frac{4(a+\sum_{k=1}^{n} x_k^T L(A) x_k)^2}{T^2\ln(2)} \\ + \frac{4}{T}\left[\log_2(T) - H(aG+gA)\right]\left[\sum_{k=1}^{n} x_k^T L(A)\frac{\partial x_k}{\partial G_{\alpha\beta}}\right] \\ - \frac{1}{T}\sum_{k=1}^{n}\left[\frac{\left[a(x_{\alpha k}-x_{\beta k})^2 + 2x_k^T L(A)x_k\right]\left[a(x_{ik}-x_{jk})^2 + 2x_k^T L(A)x_k\right]}{\lambda_k \ln(2)}\right. \\ \left. + \log_2(\lambda_k)\left[4x_k^T L(A)\frac{\partial x_k}{\partial G_{\alpha\beta}} + 2a(x_{ik}-x_{jk})\left(\frac{\partial x_{ik}}{\partial G_{\alpha\beta}} - \frac{\partial x_{jk}}{\partial G_{\alpha\beta}}\right)\right]\right]$$

where we have use the eigenvector derivatives as calculated according to Eq C.4.

Bibliography

[1] Vladimir Gligorijević and Nataša Pržulj. Methods for biological data integration: perspectives and challenges. *Journal of the Royal Society Interface*, 12(112):20150571, 2015.

[2] Wenyuan Li, Shihua Zhang, Chun-Chi Liu, and Xianghong Jasmine Zhou. Identifying multi-layer gene regulatory modules from multi-dimensional genomic data. *Bioinformatics*, 28(19):2458–2466, 2012.

[3] Chen Meng, Bernhard Kuster, Aedín C Culhane, and Amin Moghaddas Gholami. A multivariate approach to the integration of multi-omics datasets. *BMC bioinformatics*, 15(1):162, 2014.

[4] Paul Kirk, Jim E Griffin, Richard S Savage, Zoubin Ghahramani, and David L Wild. Bayesian correlated clustering to integrate multiple datasets. *Bioinformatics*, 28(24):3290–3297, 2012.

[5] Jemila S Hamid, Pingzhao Hu, Nicole M Roslin, Vicki Ling, Celia MT Greenwood, and Joseph Beyene. Data integration in genetics and genomics: methods and challenges. *Human genomics and proteomics: HGP*, 2009, 2009.

[6] Ashley J Vargas and Curtis C Harris. Biomarker development in the precision medicine era: lung cancer as a case study. *Nature Reviews Cancer*, 16(8):525, 2016.

[7] David Gomez-Cabrero, Imad Abugessaisa, Dieter Maier, Andrew Teschendorff, Matthias Merkenschlager, Andreas Gisel, Esteban Ballestar, Erik Bongcam-Rudloff, Ana Conesa, and Jesper Tegnér. Data integration in the era of omics: current and future challenges. *BMC systems biology*, 8(2):I1, 2014.

[8] Kim-Anh Lê Cao, Ignacio González, and Sébastien Déjean. integromics: an r package to unravel relationships between two omics datasets. *Bioinformatics*, 25(21):2855–2856, 2009.

[9] Ettore Mosca and Luciano Milanesi. Network-based analysis of omics with multi-objective optimization. *Molecular BioSystems*, 9(12):2971–2980, 2013.

[10] Yuanhua Liu, Valentina Devescovi, Suning Chen, and Christine Nardini. Multilevel omic data integration in cancer cell lines: advanced annotation and emergent properties. *BMC systems biology*, 7(1):14, 2013.

[11] Marylyn D Ritchie, Emily R Holzinger, Ruowang Li, Sarah A Pendergrass, and Dokyoon Kim. Methods of integrating data to uncover genotype-phenotype interactions. *Nature reviews. Genetics*, 16(2):85, 2015.

[12] Ronglai Shen, Adam B Olshen, and Marc Ladanyi. Integrative clustering of multiple genomic data types using a joint latent variable model with application to breast and lung cancer subtype analysis. *Bioinformatics*, 25(22):2906–2912, 2009.

[13] Manlio De Domenico, Vincenzo Nicosia, Alexandre Arenas, and Vito Latora. Structural reducibility of multilayer networks. *Nature communications*, 6:6864, 2015.

[14] Bo Wang, Aziz M Mezlini, Feyyaz Demir, Marc Fiume, Zhuowen Tu, Michael Brudno, Benjamin Haibe-Kains, and Anna Goldenberg. Similarity network fusion for aggregating data types on a genomic scale. *Nature methods*, 11(3):333–337, 2014.

[15] Sijia Huang, Kumardeep Chaudhary, and Lana X Garmire. More is better: recent progress in multi-omics data integration methods. *Frontiers in genetics*, 8:84, 2017.

[16] Andrew R Joyce and Bernhard Ø Palsson. The model organism as a system: integrating'omics' data sets. *Nature reviews. Molecular cell biology*, 7(3):198, 2006.

[17] Bonnie Berger, Jian Peng, and Mona Singh. Computational solutions for omics data. *Nature reviews. Genetics*, 14(5):333, 2013.

[18] Angela Serra, Michele Fratello, Vittorio Fortino, Giancarlo Raiconi, Roberto Tagliaferri, and Dario Greco. Mvda: a multi-view genomic data integration methodology. *BMC bioinformatics*, 16(1):261, 2015.

[19] Yinyin Yuan, Richard S Savage, and Florian Markowetz. Patient-specific data fusion defines prognostic cancer subtypes. *PLoS computational biology*, 7(10):e1002227, 2011.

[20] Vasileios Lapatas, Michalis Stefanidakis, Rafael C Jimenez, Allegra Via, and Maria Victoria Schneider. Data integration in biological research: an overview. *Journal of Biological Research-Thessaloniki*, 22(1):9, 2015.

[21] Derek Greene and Pádraig Cunningham. A matrix factorization approach for integrating multiple data views. *Machine Learning and Knowledge Discovery in Databases*, pages 423–438, 2009.

[22] Matteo Bersanelli, Ettore Mosca, Daniel Remondini, Enrico Giampieri, Claudia Sala, Gastone Castellani, and Luciano Milanesi. Methods for the integration of

multi-omics data: mathematical aspects. *BMC bioinformatics*, 17(Suppl 2):15, 2016.

[23] Jonathan R Pollack, Therese Sørlie, Charles M Perou, Christian A Rees, Stefanie S Jeffrey, Per E Lonning, Robert Tibshirani, David Botstein, Anne-Lise Børresen-Dale, and Patrick O Brown. Microarray analysis reveals a major direct role of dna copy number alteration in the transcriptional program of human breast tumors. *Proceedings of the National Academy of Sciences*, 99(20):12963–12968, 2002.

[24] Tero Aittokallio and Benno Schwikowski. Graph-based methods for analysing networks in cell biology. *Briefings in bioinformatics*, 7(3):243–255, 2006.

[25] Ignacio González, Sébastien Déjean, Pascal GP Martin, Olivier Gonçalves, Philippe Besse, and Alain Baccini. Highlighting relationships between heterogeneous biological data through graphical displays based on regularized canonical correlation analysis. *Journal of Biological Systems*, 17(02):173–199, 2009.

[26] Harold Hotelling. Relations between two sets of variates. *Biometrika*, 28(3/4):321–377, 1936.

[27] Kim-Anh Lê Cao, Debra Rossouw, Christele Robert-Granié, and Philippe Besse. A sparse pls for variable selection when integrating omics data. *Statistical applications in genetics and molecular biology*, 7(1), 2008.

[28] Kim-Anh Lê Cao, Pascal GP Martin, Christèle Robert-Granié, and Philippe Besse. Sparse canonical methods for biological data integration: application to a cross-platform study. *BMC bioinformatics*, 10(1):34, 2009.

[29] Richard S Savage, Zoubin Ghahramani, Jim E Griffin, Bernard J De La Cruz, and David L Wild. Discovering transcriptional modules by bayesian data integration. *Bioinformatics*, 26(12):i158–i167, 2010.

[30] Giulia Menichetti, Daniel Remondini, Pietro Panzarasa, Raúl J Mondragón, and Ginestra Bianconi. Weighted multiplex networks. *PloS one*, 9(6):e97857, 2014.

[31] Giulia Menichetti, Daniel Remondini, and Ginestra Bianconi. Correlations between weights and overlap in ensembles of weighted multiplex networks. *Physical Review E*, 90(6):062817, 2014.

[32] Claude E Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[33] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[34] Richard Chace Tolman. *The principles of statistical mechanics*. Courier Corporation, 1938.

[35] Samuel L Braunstein, Sibasish Ghosh, and Simone Severini. The laplacian of a graph as a density matrix: a basic combinatorial approach to separability of mixed states. *Annals of Combinatorics*, 10(3):291–317, 2006.

[36] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(Oct):2837–2854, 2010.

[37] Yiyu Yao. Information-theoretic measures for knowledge discovery and data mining. In *Entropy Measures, Maximum Entropy Principle and Emerging Applications*, pages 115–136. Springer, 2003.

[38] Kartik Anand, Ginestra Bianconi, and Simone Severini. Shannon and von neumann entropy of random networks with heterogeneous expected degree. *Physical Review E*, 83(3):036109, 2011.

[39] Manlio De Domenico and Jacob Biamonte. Spectral entropies as information-theoretic tools for complex network comparison. *Physical Review X*, 6(4):041062, 2016.

[40] Jacob Biamonte, Mauro Faccin, and Manlio De Domenico. Complex networks: from classical to quantum. *arXiv preprint arXiv:1702.08459*, 2017.

[41] Manlio De Domenico, Albert Solé-Ribalta, Emanuele Cozzo, Mikko Kivelä, Yamir Moreno, Mason A Porter, Sergio Gómez, and Alex Arenas. Mathematical formulation of multilayer networks. *Physical Review X*, 3(4):041022, 2013.

[42] Steven M Pincus. Approximate entropy as a measure of system complexity. *Proceedings of the National Academy of Sciences*, 88(6):2297–2301, 1991.

[43] Laura C Carpi, Osvaldo A Rosso, Patricia M Saco, and Martín Gómez Ravetti. Analyzing complex networks evolution through information theory quantifiers. *Physics Letters A*, 375(4):801–804, 2011.

[44] Tiago A Schieber, Laura Carpi, Albert Díaz-Guilera, Panos M Pardalos, Cristina Masoller, and Martín G Ravetti. Quantification of network structural dissimilarities. *Nature communications*, 8:13928, 2017.

[45] David Gfeller, Jean-Cédric Chappelier, and Paolo De Los Rios. Finding instabilities in the community structure of complex networks. *Physical Review E*, 72(5):056135, 2005.

[46] Andrew E Teschendorff and Simone Severini. Increased entropy of signal transduction in the cancer metastasis phenotype. *BMC systems biology*, 4(1):104, 2010.

[47] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[48] Manlio De Domenico, Shuntaro Sasai, and Alex Arenas. Mapping multiplex hubs in human functional brain networks. *Frontiers in neuroscience*, 10, 2016.

[49] Barry Bentley, Robyn Branicky, Christopher L Barnes, Yee Lian Chew, Eviatar Yemini, Edward T Bullmore, Petra E Vértes, and William R Schafer. The multilayer connectome of caenorhabditis elegans. *PLoS computational biology*, 12(12):e1005283, 2016.

[50] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P Gleeson, Yamir Moreno, and Mason A Porter. Multilayer networks. *Journal of complex networks*, 2(3):203–271, 2014.

[51] Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. Biogrid: a general repository for interaction datasets. *Nucleic acids research*, 34(suppl_1):D535–D539, 2006.

[52] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

[53] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[54] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[55] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[56] David W. Hosmer, Stanley Lemeshow, and Susanne May. *Applied Survival Analysis: Regression Modeling of Time to Event Data*. Wiley-Interscience, New York, NY, USA, 2nd edition, 2008.

[57] Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B Altman. Missing value estimation methods for dna microarrays. *Bioinformatics*, 17(6):520–525, 2001.

[58] Phillip Bonacich. Some unique properties of eigenvector centrality. *Social networks*, 29(4):555–564, 2007.

[59] Leo Spizzirri. Justification and application of eigenvector centrality. *Algebra in Geography: Eigenvectors of Network*, 2011.

[60] Carl D Meyer. *Matrix analysis and applied linear algebra*, volume 2. Siam, 2000.

[61] Jop Briët and Peter Harremoës. Properties of classical and quantum jensen-shannon divergence. *Physical review A*, 79(5):052311, 2009.

[62] Pedro Lamberti, Ana Majtey, Antoni Borras, Montserrat Casas, and Angel Plastino. Metric character of the quantum jensen-shannon divergence. *Physical Review A*, 77(5):052311, 2008.

[63] Ana Majtey, Pedro Lamberti, and Domingo Prato. Jensen-shannon divergence as a measure of distinguishability between mixed quantum states. *Physical Review A*, 72(5):052310, 2005.

[64] Benjamin Schumacher and Michael D Westmoreland. Relative entropy in quantum information theory. *Contemporary Mathematics*, 305:265–290, 2002.

[65] Mary Beth Ruskai. Inequalities for quantum entropy: A review with conditions for equality. *Journal of Mathematical Physics*, 43(9):4358–4375, 2002.

[66] Eric Carlen. Trace inequalities and quantum entropy: an introductory course. *Entropy and the quantum*, 529:73–140, 2010.

[67] Luca Rossi, Andrea Torsello, Edwin R Hancock, and Richard C Wilson. Characterizing graph symmetries through quantum jensen-shannon divergence. *Physical Review E*, 88(3):032806, 2013.

[68] Bojan Mohar, Y Alavi, G Chartrand, and OR Oellermann. The laplacian spectrum of graphs. *Graph theory, combinatorics, and applications*, 2(871-898):12, 1991.

[69] Fan RK Chung. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[70] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*, pages 585–591, 2002.

[71] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[72] Andries E Brouwer and Willem H Haemers. *Spectra of graphs*. Springer Science & Business Media, 2011.

[73] Chris Godsil and Gordon F Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2013.

[74] Daniel A Spielman. Spectral graph theory lecture notes. http://www.cs.yale.edu/homes/spielman/561/2009/lect02-09.pdf, 2009.

[75] Daniel A Spielman. Spectral graph theory and its applications. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 29–38. IEEE, 2007.

[76] Béla Bollobás. *Modern graph theory*, volume 184. Springer Science & Business Media, 2013.

[77] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[78] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[79] Cheng Ye, Richard C Wilson, César H Comin, Luciano da F Costa, and Edwin R Hancock. Approximate von neumann entropy for directed graphs. *Physical Review E*, 89(5):052804, 2014.

[80] Jorge Nocedal and Stephen J Wright. *Numerical optimization 2nd*. Springer, 2006.

[81] Jan R Magnus. On differentiating eigenvalues and eigenvectors. *Econometric Theory*, 1(2):179–191, 1985.

[82] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[83] Iridis 4 community pages. https://hpc.soton.ac.uk/community/projects/iridis/wiki/Iridis_4_Hardware, August 2018.

[84] Ivan Dokmanic, Reza Parhizkar, Juri Ranieri, and Martin Vetterli. Euclidean distance matrices: essential theory, algorithms, and applications. *IEEE Signal Processing Magazine*, 32(6):12–30, 2015.

[85] David G Kleinbaum. Survival analysis, a self-learning text. *Biometrical Journal*, 40(1):107–108, 1998.

[86] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(Oct):2837–2854, 2010.

[87] Allen Hatcher. *Algebraic topology*. Cambridge University Press, 2001.

[88] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lotfi Zadeh. *Feature Extraction: Foundations and Applications*. Springer-Verlag Berlin Heidelberg, 2006.

[89] RL Fox and MP Kapoor. Rates of change of eigenvalues and eigenvectors. *AIAA journal*, 6(12):2426–2429, 1968.

[90] Durbha V Murthy and Raphael T Haftka. Derivatives of eigenvalues and eigenvectors of a general complex matrix. *International Journal for Numerical Methods in Engineering*, 26(2):293–311, 1988.