Identification of gene modules using a generative model for relational data

by

Gustavo Lacerda

B.S. Mathematics, Bucknell University, 2001

M.Sc. Logic, Universiteit van Amsterdam, 2005

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University Of British Columbia

(Vancouver)

August 2010

Abstract

This thesis presents generative models for two types of relational data (edge rankings and networks) from block structures, based on Brumm et al (2008) [1]. We design and implement a set of inference algorithms, and evaluate them on real as well as simulated data. We conclude with a discussion of how well the real data fits the model, and with various extensions of the basic model.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I am thankful for Timothy Au-Yeung's extensive help in pre-processing *and* post-processing the data: generating inputs for the SOBA software, and generating plots of all kinds. This thesis would not have been ready today without him; for Wyeth Wasserman's comments and proposed edits on the biological content; and obviously to my supervisor Jennifer Bryan for giving me this project, for many many hours of discussions, and many of the ideas that shaped this thesis. Thanks to Joel Friedman for allowing me to work on this trans-departmental project, and for his role in the thesis committee.

Finally, I thank my parents, Paulo and Gracita, for their encouragement and support throughout the years.

Chapter 1

Introduction: the biological problem

1.1 The basics of molecular biology and gene-knockout experiments

One basic fact of molecular biology is that bits of DNA, known as genes, encode molecules, known as proteins, through a process of transcription and translation, which has become known as the "central dogma of molecular biology" [2] Proteins, in turn, work together to form protein complexes, which accomplish a cellular function, such as cellular transport.

These functions are then manifested in phenotypes (roughly, "visible traits"), where the most fundamental and integrated phenotype is obviously "fitness" or ability to reproduce. When two proteins work together, we say that their respective genes belong to the same "module" or "block". We regard gene modules as the idealized biological target of our modeling efforts, detailed in later chapters, which are driven by a mathematical object we call a "block structure" (i.e. a partition of the set of genes).

For a variety of reasons, biologists are interested in knowing which genes work together in modules. Now, much, if not most of molecular biology follows the principles of reverse engineering, especially: that the easiest way to figure out how

a machine works is to try to break it.

Central to this reverse-engineering approach are so-called gene-knockout experiments: by disabling single genes using genetic techniques that can replace the relevant stretch of DNA with some altered variant or even with something completely inert.

A mutated gene will, in most cases, either produce a broken protein or no protein at all (otherwise, the gene cannot be said to be "broken"). Colonies of these deletion mutants can then be placed on agar plates for direct physical observations of their phenotype. If the protein has a functional role in a protein complex and there is not a redundant gene performing an overlapping function, the mutation will quantitatively disrupt the function of the complex.

Consider two genes that encode proteins belonging to the same protein complex. A deletion mutant for either of these genes will be lacking a member of this complex and the protein complex should lose some or all functionality. By observing two such mutants in a variety of conditions, one should see a high degree of similarity in their properties (they are broken in similar ways, and will have similar vulnerabilities), compared to pairs of organisms in which the mutated genes are involved in different complexes (they are broken in different ways).

1.2 Gene profiles

Diverse phenotypes can be measured for an organism, such as growth rates (i.e. cell reproduction rates) or the capacity of a cell to metabolize a chemical in colour-based assays. Measurements of knockout mutant organisms are typically made under many experimental conditions, such as ranges of temperature, salinity or the presence of chemicals. The collection of these measurements for a given gene can be represented as a numeric vector hereafter referred to as the "gene profile".

Therefore, it is plausible that groups of genes with similar gene profiles will have very high overlap with groups of genes that encode proteins that physically interact to form a functional complex.

In addition to the quantitative phenotype measurements described, there are other quantitative gene-related measurements that can be compared in a similar manner, such as gene expression profiles (i.e. RNA or protein concentrations).

1.3 Analyzing profiles, clustering

Measures of similarity between pairs of genes have been used as inputs to hierarchical clustering (which is explained in Ch. 3), but rather than partition the set of genes into a block structure, such algorithms produce dendrograms (i.e. binary trees).

There are approaches based on thresholding the similarity values, but besides not being very principled, they have many drawbacks and practical problems, such as not using all the data, and the difficulty of choosing a threshold [1].

This thesis is based on a probabilistic model for edge rankings known as SOBA, first described in Brumm (2008) "Finding Functional Groups of genes using pairwise relational data: methods and applications", (PhD thesis, [1]). Our contribution is redefining the SOBA model to be more consistent with standard Bayesian methodology, defining search spaces, adapting search algorithms to our problem, evaluating them on real as well as simulated data, discussing the shortcomings and possible extensions of the algorithms as well as the SOBA model, and making our experiments reproducible and available as R code.

1.4 Structure of the thesis

This chapter presented the biological foundations of this work. In Ch. 2, we present a new, fully generative formulation of the SOBA probabilistic model, including edge ranking *and* network data. In Ch. 3, we present our inference technologies, all based on structure search: from computationally-cheap point estimates to expensive approximations of the posterior, and a few other options in between. In Ch. 4, we introduce our data sources, evaluation measures, and show the results of experiments on real and simulated data, with minimal interpretation. In Ch. 5, we discuss the findings of Ch. 4 in detail, and discuss possible extensions of the SOBA model and inference algorithms.

Chapter 2

The model

In this chapter, we introduce our probabilistic model for generating two types of data from block structures, namely (a) edge rankings, and (b) networks; and show expressions for the posterior distribution conditional on one or both types of data. This provides the basis for inferring an block structure based on data commonly available in genomics studies.

In this chapter, we are going to introduce generative models for edge rankings and network data. There is a large amount of genomics data that can be viewed in these forms.

2.1 Block structures and notation

Our main goal in this thesis is to use observed genomic data to infer an underlying block structure. Here we introduce some useful vocabulary and notation. Note that we use the terms "module" and "block" interchangeably and that we may occasionally use the words "cluster" or "group" or "color" with the same meaning.

Let $G = \{g_1, ..., g_n\}$ be an ordered set of n nodes. A block structure B on G is simply a partition on G (in the set-theoretic sense), i.e. a set $B = \{b_1, ..., b_k\}$ of blocks such that for all $i \neq j$, $b_i \cap b_j = \emptyset$ and $\bigcup_i b_i = G$. The set of all block structures on G is written $\mathbb{B}_G$. By convention, we use $n = |G|$ to be the number of nodes, $N = n(n-1)/2$ to be the number of edges, and $k = |B|$ to be the number of blocks.

Labeling representation: We can represent a block structure by assigning a unique capital letter to each block. For example "AABB" represents the block structure $\{\{g_1, g_2\}, \{g_3, g_4\}\}$. However, note that "BBAA" also represents the same structure, and that in general, any permutation of the labels will as well, leading to the conclusion that there are $k!$ labelings for each block structure.

Normal form labeling: a labeling is in normal form if the block labels are sorted alphabetically. This means that out of all labelings that represent B, only the first (in lexicographic ordering) is in normal form. This means in particular that the letter assigned to the first node is "A", that the first node appearing in a different block will be assigned a "B", and so forth.

The Normal Form Labeling of block structure B is denoted $NFL(B)$, represented as a vector of labels. The fact stated above is that for all B, $NFL(B)_1 =$"A". As a shorthand, we write $NFL(B)_i$ as $l_B(g_i)$, i.e. the label of g_i.

2.2 Review of Bayes' rule

Bayes' rule states that: $P(h = h_0|D = D_0) = \frac{P(D=D_0|h=h_0)P(h=h_0)}{P(D=D_0)}$, where typically h is the hypothesis, and D is the data. The left-hand side is called the **posterior probability**. $P(h = h_0)$ is called the **prior probability** of h_0, and $P(D = D_0|h = h_0)$ is called the **likelihood** of the data, which is a function of the hypothesis h. It is standard to remove the equalities, thus abbreviating $P(D = D_0)$ to $P(D)$, and so forth.

The standard form of Bayes' Rule is: $P(h|D) = \frac{P(D|h)P(h)}{P(D)}$.

2.3 Ranked relational data

Consider the empty graph on the set of nodes G and imagine the edges arriving one by one, until the complete graph is obtained. The ordered list of edges is called an "edge ranking", and denoted by a vector π, where π_1 refers to the top-ranked edge, i.e. the "first arrival" and so on. The basic idea of our generative model is that edges between nodes in the same block will tend to arrive before – or have better/higher ranks – than edges between nodes in different blocks. Given the block structure B and for any pair of nodes g_i and g_j, we define a binary-valued function "within": $w_B(g_i, g_j)$ that returns 1 when g_i and g_j are in the same block, i.e. $l_B(g_i) = l_B(g_j)$,

and 0 otherwise. We are interested in using an observed edge ranking to discover the latent block structure into which the nodes are organized. Informally, the block structure that best explains the observed edge ranking data, i.e. that best upholds the assumption that "within" edges arrive before "between" edges, will be our best estimate of the truth. In the remainder of this thesis, we will drop the quotes, and use a hyphen instead: "within-edge" and "between-edge".

Stage-wise Edge Ranking Models specify generative distributions over edge rankings using edge strengths, whose values are positive real numbers: at any given point in the edge-adding process, the probability of a given edge being the next selection is its strength divided by the total strength of the remaining edges.

This work focuses on a subclass of the above, Homogeneous Ranked Stochastic Block Models (RSBM) [1], in which the edge strengths $\lambda(e)$ are determined by a block structure together with a single parameter $r > 1$: within-edges have strength r, whereas between-edges have strength 1. For the rest of this thesis, we call this "the SOBA model".

When $r = 1$, edge ranking data is uninformative about the underlying block structure. At the other extreme, as $r \to \infty$, the edge ranking data will lead us to the true block structure every time.

At any step in the random graph process, the probability of selecting edge e is:

$$P(e|B,S) = \frac{\lambda_B(e)}{\sum_{e' \in S} \lambda_B(e')} \tag{2.1}$$

where B is the true block structure, $\lambda_B(e)$ is e's strength according to B (either r or 1), and S is the set of remaining edges.

Recall that $N = n(n-1)/2$ is the number of edges. Then the edge ranking likelihood is:

$$P(\pi|B) = P(\pi_1|B, S = \{\pi_1, ..., \pi_N\}) P(\pi_2|B, S = \{\pi_2, ..., \pi_N\}) ... P(\pi_N|B, S = \{\pi_N\}) \tag{2.2}$$

(the last factor reduces to 1, since it is the probability that the next edge is π_N given that the only edge remaining is π_N.)

When working with real data, we obtain edge rankings by ranking direct and/or indirect measurements on gene-gene relationships. However, it is also possible to simulate edge rankings from a user-specified block structure and we do so now to illustrate the meaning of the strength parameter r. Fig. 2.1 shows a sample of the graph process when $r = 20$. Node color encodes block membership.

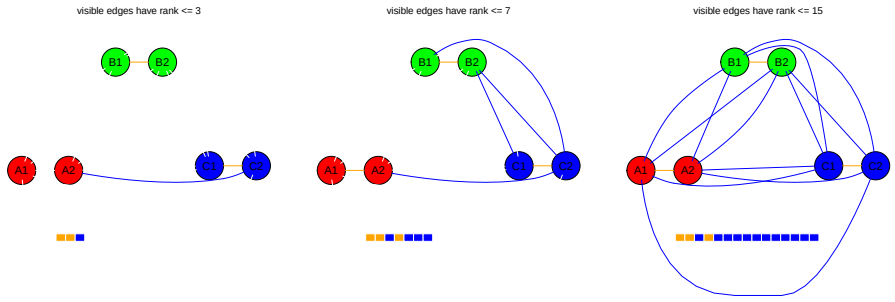

Figure 2.1: Edge arrivals – frames from the animation of a random sample of a [2,2,2] model when $r = 20$.

Fig. 2.2 depicts some simulated edge rankings for different strength ratios, where within-edges (colored orange) tend to arrive earlier than between-edges (colored blue) and that this becomes more pronounced as r becomes large. For each value of r, we generated 50 samples from a ground truth containing 3 modules, each one containing 2 nodes (we say that B is of type [2,2,2], which implies 3 "within" and 12 between-edges).

The edge arrivals are shown from left (first) to right (last). We use orange to represent within-edges, and blue to represent between-edges. We can see that the bigger the r, the more the orange concentrates on the left side. Each row depicts a single observed edge ranking, which can also be visualized as a sequence of graphs in which one edge is added at every step, as in Fig. 2.1.

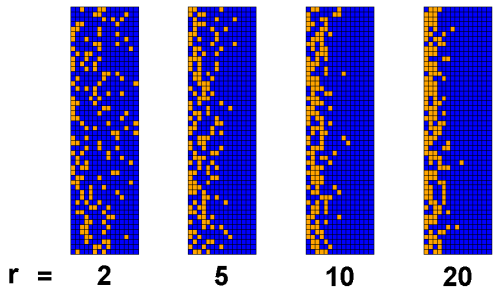

Figure 2.2: Heatmap visualization of edge rankings.

Taking a Bayesian estimation approach to do inference on the block structure B, the posterior distribution given an observed edge ranking π is given by:

$$P(B|\pi) = \frac{P(\pi|B)P(B)}{P(\pi)} \tag{2.3}$$

$$= \frac{P(\pi|B)P(B)}{\sum_{b\in\mathbb{B}} P(\pi|b)P(b)} \tag{2.4}$$

As we will see in the next chapter, the sum in the denominator is intractable, but can be estimated (lower-bounded) by summing over a set of high-scoring models.

2.4 Network data

Block structures can also give rise to so-called network data. A network A is a set of edges between pairs of nodes in G, typically a sparse one. We will define a generative model for network data based on a block structure where the essential idea is that the probability of an edge existing between a pair of genes in the same block is greater than for pairs where the genes are in different blocks.

Given a block structure B, a generative model for the network A is given by:

$$\gamma = P(e_A(g_i, g_j) = 1 | w_B(g_i, g_j) = 1) \tag{2.5}$$

$$\delta = P(e_A(g_i, g_j) = 1 | w_B(g_i, g_j) = 0) \tag{2.6}$$

where $e_A(g_i, g_j)$ is a binary-valued function that returns 1 when the edge $(g_i, g_j) \in A$, and 0 otherwise and the parameters γ and δ represent the probabilities of edge presence conditional on co-membership and non-co-membership, respectively. If $\gamma = 1$ and $\delta = 0$, then the network data by itself will always give us perfect performance in the task of finding a block structure. At the other extreme, if $\gamma = \delta$, then the network data is useless.

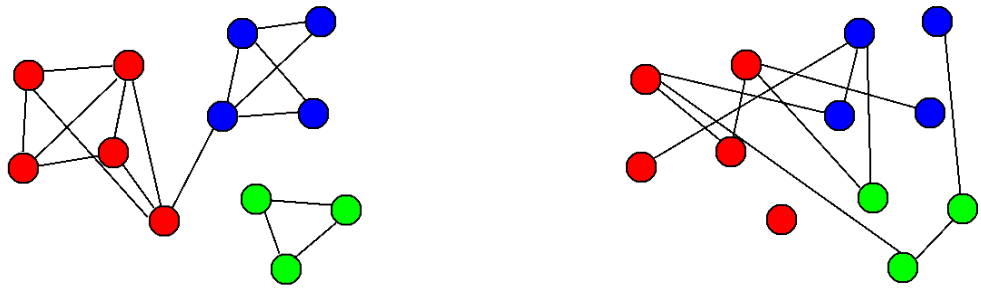

Figure 2.3: Simulated network data for different values of (γ, δ): (a) $\gamma \approx 0.85$, $\delta \approx 0.02$: modules appear clearly (b) $\gamma = \delta$: network data is useless noise.

For block structure B and network A, it is useful to count the number of edges in (not in) A that are "within" and "between" edges, with respect to B:

$$\begin{aligned} s_{11} &= \#\{i < j : w_B(g_i, g_j) = 1, e_A(g_i, g_j) = 1\} \\ s_{10} &= \#\{i < j : w_B(g_i, g_j) = 1, e_A(g_i, g_j) = 0\} \\ s_{01} &= \#\{i < j : w_B(g_i, g_j) = 0, e_A(g_i, g_j) = 1\} \\ s_{00} &= \#\{i < j : w_B(g_i, g_j) = 0, e_A(g_i, g_j) = 0\} \end{aligned} \tag{2.7}$$

These counts are sufficient statistics for the above model, and they are what we use when doing inference, as below. The network likelihood is:

$$P(A|B) = \gamma^{s_{11}}(1-\gamma)^{s_{10}}\delta^{s_{01}}(1-\delta)^{s_{00}} \tag{2.8}$$

And the log-likelihood:

$$logP(A|B) = s_{11}log(\gamma) + s_{10}log(1-\gamma) + s_{01}log(\delta) + s_{00}log(1-\delta) \qquad (2.9)$$

This thesis extends and modifies prior work by [1], in which network data is used to form "prior" probabilities of node pairs belonging parentheses? to the same block. Note that Brumm's usage of the word "prior" represents something of an abuse of terminology, in that it does not fulfill the strict Bayesian meaning of "a belief before seeing any data". This approach has two drawbacks: (a) it applies an asymmetric treatment of edge ranking and network data and (b) it does not support inference based on network data alone.

In this thesis, we treat the two types of data symmetrically, i.e. as if they are observed at the same time, rather than artificially assuming that the network data is observed prior to the edge ranking. This also enables us to perform inference using only network data, though in practice, this is rarely desirable given the sparseness of network data of the type we will typically use.

2.4.1 Models for missing network data

The observed network data generally comes from experimental assessments of direct physical interactions between two proteins (gene products) or genetic interactions (a more abstract concept). Since these experiments are demanding and the number of possible gene pairs is large, the number of gene pairs that are experimentally probed is generally very much less than N. Therefore, one may want to incorporate this information into the model for network data, i.e. record for each edge one of 3 possible values: {present, absent, unobserved}.

However, these experiments are not conducted "at random", rather biologists prioritize the study of gene-gene relationships that they perceive to be interesting, usually those that they expect to exist. Thus, a lack of measurement is itself evidence of an absent edge (based on scientists' prior knowledge), but not as strong as the evidence of an actual negative observation. In an extended model for network data, one could incorporate two parameters ζ and η for the probability of measurement conditional on co-membership and non-co-membership, respectively.

$$\zeta = P((i,j) \text{ is observed } | w_b(g_i, g_j) = 1) \tag{2.10}$$

$$\delta = P((i,j) \text{ is observed } | w_b(g_i, g_j) = 0) \tag{2.11}$$

In reality, negative results are not reported as diligently as positive ones and, therefore, when using publicly available data it is simply impractical to determine which relationships were tested but deemed to be absent. Therefore, in this thesis, we do not further develop the more complex model.

ABSENT: unobserved = absent

In [1], unobserved edges are treated as absent. This amounts to setting $e_a(g_i, g_j) = 0$ for all unobserved edges (g_i, g_j). This assumption could be justified in datasets for which lack of measurement is evidence of the experimenter's prior knowledge that the edge couldn't plausibly exist. The advantage of this is that it incorporates the biologists' domain knowledge; the disadvantage is that it treats missing observations on par with actual negative measurements, which is almost certainly wrong. In practice, large public datasets often do not publish which node-pairs were measured, making this assumption necessary.

Therefore, in all experiments, this is the model we used.

MAR: Missing At Random

Finally, it is interesting to contemplate the scenario where the unobserved edges are a uniformly drawn subset of the N possible edges, i.e. the "missing at random" (MAR) assumption. Using the parameters introduced above, MAR implies equality of ζ and η, the probabilites of edge observation based on block status. We would still use Eqs. 2.8 and 2.9 as data-generating models, but most of the terms in the network likelihood would be omitted, since only the observed edges are informative. This makes MAR the computationally most efficient way of dealing with missing data.

One important difference to highlight between the two previous models is that ABSENT effectively uses much more data, and is accordingly much more influen-

tial (in the sense of having a stronger effect on the total likelihood) for the same values of γ and δ. Therefore, if switching between these missing-data models, it may make sense to adjust γ and δ accordingly.

2.5 The full model

If both edge ranking and network data are available, we assume that, conditional on the block structure, they are generated independently. Thus we get a graphical model as follows:

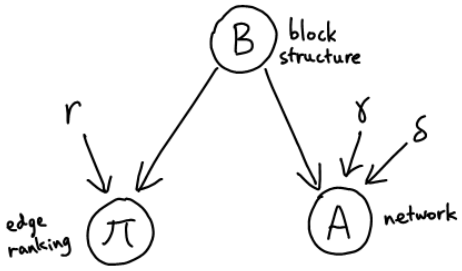

Figure 2.4: The full model, as a graphical model.

Thus:

$$P(\pi, A|B) = P(\pi|B)P(A|B) \tag{2.12}$$

The total log-likelihood thus becomes: $logP(\pi, A|B) = logP(\pi|B) + logP(A|B)$
The total posterior is:

$$P(B|\pi, A) = \frac{P(B)P(\pi, A|B)}{P(\pi, A)} \tag{2.13}$$

$$= \frac{P(B)P(\pi, A|B)}{\sum P(B)P(\pi, A|B)} \tag{2.14}$$

The weighting of the different types of data is dependent on the parameters (r for the edge ranking data; γ and δ for the network data). Ideally, they should be estimated from data. However, in this thesis, we run inference with a priori plausible values, in a somewhat ad-hoc fashion.

Again, the sum in the denominator is intractable, and in the next chapter we will see how to estimate (lower-bound) it with a set of high-probability structures.

Chapter 3

Block structure inference

In this chapter, we present ideas and algorithms for Bayesian inference on block structures, i.e. inference of posterior distributions over block structures B, and point estimates aiming to maximize the posterior probability. Faced with the choice between quality and computational cost, we present a set of inference methods at different points along this trade-off.

3.1 The hierarchy of inference algorithms

Ideally, one would like to obtain the full posterior distribution for the block structure. This would allow us to answer targeted, biologically interesting questions, such as "What is the posterior probability that the genes VPS55 and VPS68 are in the same block?". With knowledge of the full posterior, Bayesian averaging allows us to answer this question by looking at the posterior mass of all block structures that place these two genes in the same block relative to the total posterior mass. However, for moderate n, a complete representation of the posterior distribution would involve too many block structures [1].

Therefore, one can resort to making an **approximation of the full posterior**, by searching for a set of high-scoring models and using that set as a summary of the

[1] When k is known, the number of possible block structures is $S(n,k)$, where S refers to the Stirling numbers of the second kind. When k is unknown, the number of structure is the nth Bell number. Both of these grow very rapidly! The 10th Bell number is larger than 10^6, the 20th is greater than 5×10^{13}

posterior distribution (using an algorithm such as **MOSS**), hoping that they capture most of the probability mass therein. However, even for moderate n this can still be too expensive and we must be satisfied with less information about the posterior.

A natural, cheaper alternative is to do **MAP inference** (maximum a posteriori), which means finding the point estimate corresponding to the single best-scoring structure, a.k.a. best mode or highest peak of the posterior objective. Although the MAP estimate provides a simple summary and is less costly to compute, it does not represent our uncertainty in the estimate. To return to the probability of co-membership example from above, we note that the MAP estimate can only guess at this with a posterior probability of 0 or 1. Nonetheless, in many practical situations, the MAP estimate is still extremely useful. Unfortunately, when $n > 50$, even this becomes expensive, and we instead obtain an estimated block structure through **score-based dendrogram-pruning**, in which we use our posterior mass (or, equivalently, our likelihood) as an objective function to maximize over the space of dendrogram cuts.

Cluster analysis and dendrogram representations thereof, described below, are already heavily used by the genomics community for visualizing relationships between genes and assessing the existence of distinct gene groups or clusters. Therefore, it is very natural for us to adapt our methods in large n settings to search for high-scoring block structures within those that are obtainable by pruning a dendrogram.

These inference methods are shown in Fig. 3.1 and described in this chapter. We introduced the methods in the order from most sophisticated (and most costly) to simplest (and fastest). But because each of these inference methods can be used to inform (via initialization) a more sophisticated one, they will now be presented in the reverse order.

3.2 Dendrograms

A dendrogram (see Fig. 3.2) on n objects is a rooted binary tree with $n-1$ levels. The most common method of forming a dendrogram based on observed similarities between pairs of objects is an iterative, bottom-up procedure called agglomerative hierarchical clustering. Initially, each of the n objects is a cluster of size 1. The

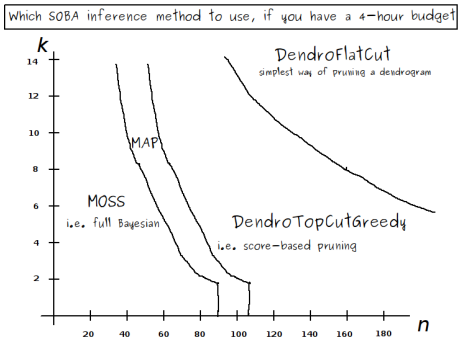

Figure 3.1: Cartoon plot – when it's feasible to use each inference method. This is based on informal experience and analysis. It is assumed that MOSS has a constant factor more evaluations than MAP. See "Complexity" section for details on the shapes of these curves.

pair of clusters that are most similar are then merged to form a new cluster. This procedure is repeated $n-1$ times, corresponding to the levels in the dendrogram, until all objects belong to one cluster of size n. Since most of these merge events occur between two clusters that contain many individual objects, it is necessary to specify how the distance between two clusters is computed from the pairwise distances between their constituent objects. The three most common choices are to take the minimum, the average, or maximum distance, respectively, leading to single, average, and complete linkage clustering. Of these, average is by far the most common method in practice, especially in genomics. We use the R function `hclust` to implement average linkage hierarchical clustering and provide ranked relational data for the object-to-object similarities.

Block structures can be obtained by **cutting** (a.k.a. pruning) a dendrogram, i.e. removing edges and taking the connected components as blocks. Dendrogram cuts will be used in two ways: (a) as a space of possible structures to evaluate (a more tractable alternative to the space of all structures); and (b) in structure searches like MAP and MOSS: as a smarter alternative to random starting points.

3.2.1 Dendrogram cuts

Given that we desire to use a dendrogram to get a block structure with k blocks, here are some notions of dendrogram cuts:

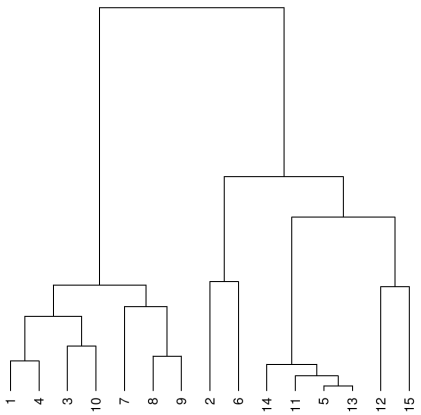

Figure 3.2: Dendrogram of 15 genes from the Schluter15 dataset, produced by feeding ranked distances to hclust.

- A **flat cut** of a fully-ordered dendrogram is obtained by pruning at one fixed level. For any given k, there is a single flat cut, which corresponds to removing all merges above level $n-k$. By itself, this is a type of block structure inference from relational data, but it doesn't take advantage of the likelihood, which also incorporates network data. Flat cuts are a standard tool in bioinformatics, and we will think of them as a basic baseline that we wish to beat.

- **Top cuts** are a generalization of the notion of flat cut, in which the $k-1$ removed merges form a subtree that includes the root node. There is a bijection between the set of top cuts and the set of subtrees that include the root and contain $k-1$ nodes. Therefore, the space of top cuts also has size exponential on k, i.e. $O(2^k)$.

- **Arbitrary cuts** are obtained by removing *any* $k-1$ merges from the dendrogram. The space of arbitrary cuts has size exponential on k: $(n-1)(n-2)...(n-k)$ i.e. $O(n^k)$. [2] The notion of arbitrary cut promises to become more useful in a particular extension of the SOBA model, in which one of the k blocks is the "junk block" and edges inside it have the same weight as

[2] **Note:** removing an edge from the top of a component automatically removes the other one too.

"between" edges.

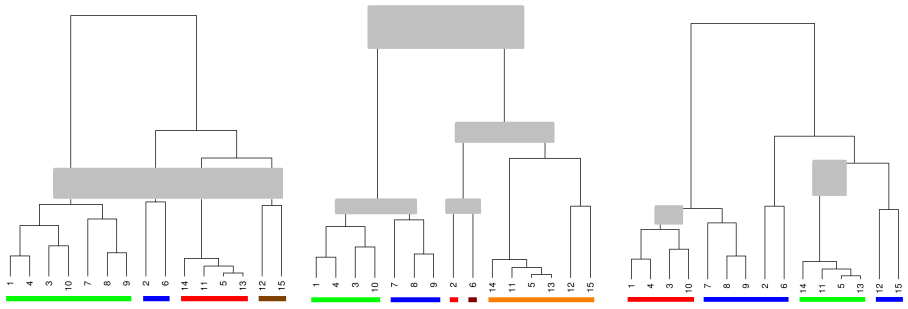

Figure 3.3: Dendrogram cuts. The grey areas indicate the removed merge events. The blocks are the connected components of the resulting branches. (a) the flat cut for $k = 4$ (b) a top cut for $k = 5$. Note that the grey blocks form a subtree containing the root node. (c) an arbitrary cut for $k = 3$.

When using top cuts or arbitrary cuts, we can compare the various structures by using our Bayesian score from Section 2, and return the one with the highest score.

Whether we're exploring the space of arbitrary cuts or top cuts, we can either do an **exhaustive search**, in which the entire space is explored (i.e. every structure in it is evaluated); or a **greedy search**, in which a new block gets split at each iteration. Since these spaces can become quite large as k gets big, a greedy search may sometimes be advisable. [3]

The SOBA software provides two inference methods using average-link clustering: `DendroTopCutGreedy` (greedy search over the space of top cuts) and `FlatCut` (flat cut).

3.2.2 Complexity

A greedy search in the space of top cuts involves $2+3+...+(k-1)$ evaluations (on the first iteration, there is only one possible split, so it doesn't need to be scored), which is $O(k^2)$ evaluations. Since each evaluation is $O(n^2)$, the cost of

[3] It is also easy to extend greedy searches to look ahead further than 1 level, and find the optimal m steps ahead.

this procedure scales as $O(n^2k^2)$, which is a lot better than $O(n^4k)$, especially since k is much smaller than n.

3.3 MAP-finding and optimization

If we have more computing resources, we can do better than dendrogram pruning, and try to obtain the point estimate known as the **MAP estimate** (Maximum a Posteriori), i.e. the highest-scoring structure, or the best mode of the posterior distribution. Obtaining the MAP estimate involves potentially hard optimization problems, but this is necessary when one is doing Full Bayes or only MAP inference.

3.3.1 Using the neighborhood

Given a notion of neighborhood, the optimization is typically done by a procedure in which, at every iteration, one moves to a better neighbor (in some cases, the best neighbor). [4]

The trouble is that when the posterior has multiple modes, one can get stuck in local maxima, and one never gets a mathematical guarantee that the **putative MAP estimate** (i.e. the best mode found), is actually the actual MAP (the best mode). Note however, that the notion of mode / local maximum is dependent on the notion of neighborhood, since we define modes as a local maxima, i.e. structures whose score is at least as high as all other structures in its single-relabeling neighborhood.

The neighborhood relation

One can imagine many possible neighborhood relations between block structures. In this thesis, we focus on one.

The **single-relabeling neighborhood** of block structure s is defined as: $SRN(s) = \bigcup\{s' : HD(s,s') = 1\}$, where HD is the Minimum Hamming Distance between all labellings of the two structures, i.e. the set of block structures obtained by changing the label of any single gene to any other label, i.e. reassigning it to a different

[4] another possibility is simulated annealing, in which, at every iteration, one also can move to worse neighbors, but only with some probability, which is a function of a *temperature* parameter. However, we do not explore this further.

block. The single-relabeling neighborhood of a block structure can have up to $n(k-1)$ structures (fewer in cases where two or more relabellings are seen to be equivalent after putting them in normal form). [5] In this thesis, since we consider k to be fixed, relabellings which change the number of blocks are not considered part of the neighborhood.

3.3.2 Initialization

When discussing optimization algorithms above, we didn't specify where to start the optimization. Here we present some options.

- **Random initialization** We initialize at a random block structure. Experiments have shown that, much of the time, the putative MAP estimate obtained this way is in fact not the real MAP estimate. However, experiments also suggest that *multiple* random restarts will usually lead to actual MAP estimate. [6]

- **Informed initialization** Running optimization many times (from multiple random starting points) is computationally costly. However, the dendrogram cuts discussed in the previous section can be good starting points that are computationally feasible [7]. The SOBA software provides the `MAP` inference method, which is initialized with a flat cut.

3.3.3 Complexity

A back-of-the-envelope analysis suggests that the computational cost of optimization tends to scale as $O(n^4k)$ or worse. The argument follows. The cost of a model evaluation scales as $O(n^2)$ (since the log-likelihood has $O(n^2)$ terms); the size of neighborhood scales as $O(n(k-1))$ and the number of steps from initialization to

[5] One can also define a merge-split neighborhood, in which neighbors are obtained by splitting a block or merging two blocks, but we do not consider it in this thesis.

[6] we have found that deeper modes are also wider (in the sense that their *attractor basins* contain more block structures, see Appendix). By starting from random block structures, one will eventually start from one that leads to the MAP, and probably sooner rather than later.

[7] Study forthcoming

convergence scales as $O(n)$. In practice, this means that it is prohibitively expensive to scale optimization methods up to 200+ genes and 10 blocks, at least when using the single-relabeling neighborhood.

3.4 Structure search and posterior approximation, and why sampling is a bad idea

If we can still afford some computation after obtaining the MAP estimate, we would like to obtain *several* high-scoring structures with which to approximate the posterior distribution (in lieu of computing the full posterior, which is unfeasible, since there are too many structures to evaluate). The thought that comes to the mind of most Bayesian practitioners at this point is to use MCMC to obtain samples from the posterior. We thus start out by going over sampling-based approaches, and why they are a bad idea in this case.

3.4.1 Sampling

Markov Chain Monte Carlo (MCMC) is a widely-known sampling-based approach to approximating posterior distributions, when no analytical form for the posterior is available. Each possible setting of the free parameters (i.e. the parameters whose distributions we are updating) is viewed as a state of a Markov Chain, whose stationary distribution encodes their posterior distribution. For a review, see Neal (1993) [5].

In Gibbs sampling, the parameters are updated one by one, by sampling from each parameter conditional on the rest. When Gibbs sampling isn't possible (e.g. because we don't know how to sample from some full conditionals) or desirable (e.g. because the posterior shape has a direction that does not align well with the parameters), one typically uses the Metropolis-Hastings (MH) algorithm, which works with a proposal distribution to potentially update all the parameters simultaneously.

The only requirement for running MH is the ability to compute the ratio between the posterior probability of two states (i.e. two settings of the free parameters; block structures in our case). This is required for computing the acceptance probability: $min(1, \frac{p(b^*)}{p(b)})$, where b is the current state and b^* is the proposed state.

Some of the time, the proposal will be rejected, and MCMC stays in the same state. This behavior is required for sampling correctly from the posterior distribution.

3.4.2 Structure search

Sampling gives inherently noisy mass estimates. Therefore, when attempting to visualize the posterior from a set of samples, one question that comes to mind is: how do we estimate the pdf of the posterior? In the case of continuous parameter spaces, we have the problem of density estimation (often solved by Gaussian kernels with shrinking bandwidth), and in discrete spaces we have a problem of "mass estimation", namely estimating a categorical distribution.

However, as is evidenced by the MH algorithm itself, we can compute posterior ratios exactly, even if we cannot compute the likelihood exactly, so it would be wrong to believe that state s_1 has more posterior mass than s_2 just because it has more samples. The "search" approach to inference (henceforth, Search) is that, rather than counting samples, a better approximation is obtained by visiting all states having non-negligible mass and remembering their probability masses relative to each other. We thus bypass the estimation step, and nothing is lost. As implied by its name, Search is only about exploring the space to find high-scoring states. Since it doesn't need to sample from any distribution, it doesn't need to visit high-scoring states multiple times, and is free to explore the space efficiently.

Finally, while MCMC typically exits after a pre-defined number of iterations, Search algorithms may only exit after satisfying conditions specifying that the search has been exhaustive enough. So while MCMC may never visit some neighbors of high-scoring models, as we will see, some Search algorithms will not exit until this condition is met.

3.4.3 Why does one *ever* use MCMC, then?

In continuous parameter spaces, it is easy to see why the idea of Search, as stated here, cannot work: there is an uncountably infinite number of models in the neighborhood of every mode, and thus any finite set of models captures a measure-zero set of the parameter space. This problem can be addressed by grid- and mesh-based

integration (which convert sets of "points" having probability density to "areas" or "volumes" having probability mass), but that is beyond the scope of this thesis. It is reported, however, that as the number of dimensions goes up, such approaches become intractable more rapidly than MCMC.[8]

We can now restrict the question to the case of discrete parameter spaces: shouldn't one always prefer Search then? Cases where we have a marginal likelihood may provide a counterexample.

The posterior over B is written as follows:

$$P(B|\pi) = \frac{P(\pi|B)P(B)}{P(\pi)} = \frac{P(\pi|B)P(B)}{\sum_{B \in \mathbb{B}_G} P(\pi|B)P(B)} \tag{3.1}$$

If the likelihood $P(\pi|B)$ is easy to compute, then we can do a Search over the possible settings of B, and bypass the estimation problems associated with sampling. The sum in the denominator is estimated as the sum over the visited values of B, rather than all values of B (so in fact, this is a lower bound).

However, if computing the likelihood involves a costly integral over a high-dimensional parameter (imagine r being a high-dimensional vector, such as an extension of SOBA in which each block has its own r):

$$P(\pi|B) = \int P(\pi|B,r)P(r|B)dr \tag{3.2}$$

then MCMC may be better.

Although this integral can be approximated by running an inner MCMC on r conditional on B, which produces samples $r^{(i)} \sim r|B$, and computing the sum $\sum_i P(\pi|r^{(i)})$; doing one run of MCMC for every iteration of the search is going to be very slow.

However, as we will see next, in the SOBA model, the only parameters we could possibly want to integrate out are r, γ and δ, but in practice, we take these values as known and no integration is necessary (and, even then, integrating 3 variables might not be so bad). In future work, we will use point estimates of these parameters, again bypassing the need for MCMC.

[8] I have heard this from more than one person, but have yet to find a reference.

3.4.4 MOSS algorithm: Mode-Oriented Stochastic Search

Dobra et al (2009) proposed the MOSS algorithm, which returns a set S of high-scoring models, which together approximate the posterior distribution.

The estimated posterior distribution is thus:

$$\widehat{P(B_0|\pi)} = \frac{P(\pi|B_0)P(B_0)}{\sum_{B\in S} P(\pi|B)P(B)} \quad \text{, if} B_0 \in S \tag{3.3}$$

$$= 0 \quad \text{, otherwise} \tag{3.4}$$

Since we cannot sum over all of $\mathbb{B}_G$, we are taking the set S as a surrogate.

After starting with a set of structures S, at each step, MOSS randomly chooses a structure $m \in S$ whose every neighbor m' will be evaluated (each of which, if successful, gets added to S). MOSS takes one parameter c', which is a ratio of probabilities, indicating how low we set the bar, relative to the top-scoring structure seen so far. [9] The lower the c', the better the resulting approximation, and the longer it will take for MOSS to terminate. When $c' = 1$, the search outputs only the (putative) MAP estimate; when $c' = 0$, MOSS only terminates once every possible block structure has been evaluated. In the SOBA software, this inference method is implemented as `MOSS(cprime)`, and it is initialized from a flat cut.

```
[S] = moss(objective, neighborhoodRelation, initialStructures, cprime)
   mark initialStructures as unexplored and let S be this set.
   incumbent <- initialStructure  ##top structure so far
   fmax <- objective(initialStructure) ##score of top structure so far
   while(there are unexplored structures in S)
     m <- a structure randomly sampled from the unexplored
structures in S
     mark m as explored
     for every m' in the neighborhood of m
       logpost <- objective(m')
       if (logpost>log(cprime)+fmax  AND m' not in S)
         add m' to S, as unexplored.
```

[9] As a metaphor, imagine an exclusive golf club, where all members must have at least 1% as much money as the richest member.

```
          if logpost>fmax
            incumbent <- m'
            fmax <- logpost
            remove from S all structures whose score
is < log(cprime) + fmax.
          end if
        end if
      end for
    end while
    return S
end function
```

3.4.5 How to make the best use of MOSS

As with MAP estimation, MOSS can easily fail to find the best mode, since the posterior tends to be multimodal with respect to the single-relabeling neighborhood (Appendix (?) shows optimizations from random starting points converging to different modes). Again, this highlights the importance of informed initialization and multiple initialization. Furthermore, it is a good idea to optimize the initial structures (using the same algorithm as MAP) before passing them to MOSS, because optimization algorithms will tend to find modes more quickly (this is what they are designed to do), whereas MOSS's exploration behavior is most useful near the modes of the distribution.

Another, less severe, way in which MOSS can fail to give good results is that it finds the best mode, but misses another good mode. This will happen when none of our initial structures are in the attraction basin of this mode and our c' isn't low enough to get to the bottom of our peak, from where other peaks can be reached. Whereas in MAP estimation, one selects the single best mode found from several starting points, in MOSS we want to initialize from multiple modes, except when c' tells us otherwise. So initializing MOSS is not as simple as plugging in the output of MAP.

Finally, when c' is too close to 1, MOSS is equivalent to MAP estimation, and fails to capture the uncertainty. Similarly, a high value of c' could mean that S

contains only 2 or 3 structures even while the posterior is very smooth. And when c' is too low, we can easily run out of computing time. One idea is to have a schedule that obtains c' guarantees successively, e.g. we can first run MOSS using $c' = 0.2$ until all models with $> 20\%$ as much mass as the putative MAP have been explored; once that returns, we continue with $c' = 0.1$, and so forth.

Fig. 3.4, we see a cartoon of the posterior landscape, plotted as if our discrete space of block structures were $\mathbb{Z}^2$. They show fictional runs of MOSS if it had a budget of exactly 20 structure evaluations before stopping. (a) starting from a random structure, (b) starting, luckily, from the best mode, (c) starting, rather unluckily, from a suboptimal mode. The horizontal lines in the illustrations below correspond to the contours in the illustration above. MOSS's c' parameter determines how low we will go before the algorithm stops, e.g. there is a value of c' that implies that all reachable shades of "dark grey" must be on the output set.

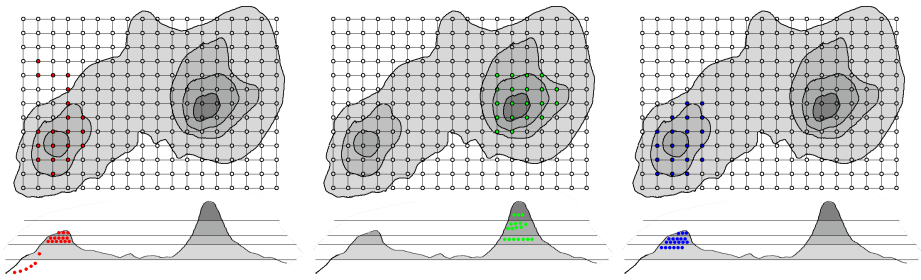

Figure 3.4: Cartoon of the posterior landscape.

Chapter 4

Empirical studies of performance

In this chapter, we introduce a real dataset from a recent study aimed at identifying protein complexes involved in endosomal transport, an essential cellular process (Schluter et al 2008, [6]). This will be our edge ranking data. Furthermore, we introduce a protein-protein interaction dataset, which we use as our network data (Kiemer et al, 2007, [4]).

Using the Schluter data and simulated data, we explore how well the point estimates described in the previous chapter (`MAP` , `DendroTopCutGreedy` , `DendroFlatCut`) perform, as measured by similarity between the estimated block structures and the underlying ground truth. Methods that return an approximation to the posterior, such as MOSS, are not evaluated here.

4.1 Introduction to the Schluter data

The Schluter data consists of gene-knockout profiles (see Ch. 1): each gene's profile is a vector of 14 numbers, where each number records a quantitative phenotype for the corresponding deletion mutant (a transgenic strain of yeast that is missing that specific gene and therefore the associated protein). There are 14 elements to the profile, because the deletion mutants are studied under many different treatments, such as stimulation by caffeine or poisoning with the antifungal amphotericin, and there are two different phenotypes of interest, growth and a measure of protein-sorting efficiency.

4.1.1 Edge-ranking data

The Schluter dataset contains profiles for 279 genes but we focus on the subset of 159 genes for which we have a high quality, though certainly imperfect, protein complex annotation. It is only for these genes that we have an acceptable notion of ground truth and, therefore, can study the performance of different inference methods.

These $n = 159$ genes are annotated to 40 protein complexes, where some large complexes have 15 to 20 constituent genes in the dataset and many others are only represented by 1 or 2 genes.

The edge ranks represent the ranks of the Euclidean distances between these profiles. In Fig. 4.1(a), we show the within-edge ranks for a sample of 30 genes drawn from the Schluter data (namely, `dataset_061`). These genes break down into 5 protein complexes, corresponding to the horizontal lines.

The ranks of within-edges for each complex are represented by dots. We see that the Schluter data seems compatible with the SOBA model, in that within-edges do tend to have better ranks than between-edges. But we also see some departures from the model. For example, the within-edges of the DNA/SWR complex seem to have better and more homogeneous ranks than those of other complexes, suggesting that complexes may have different values of "r". We also see two distinct clumps in the ranks of within-edges for the GLYCOSYL complex, which may suggest some block substructure such a protein complex with two subunits. Since neither of these observations hold to the same degree for the simulated data (Fig. 4.1(b)), we may conclude that they are characteristic of the Schluter data. This way, they point at possible generalizations of the SOBA model.

In Fig. 4.2 we show an average linkage dendrogram for the 159 complex-annotated Schluter genes and the color-coding of the gene labels reflects the annotated ground truth.

4.1.2 Network data

The network data was obtained from Kiemer et al (2007) [4], in which the set of genes investigated has substantial overlap with the annotated genes from Schluter et al (2008) [6]: of the 159 annotated Schluter genes, 92 appear on this dataset.

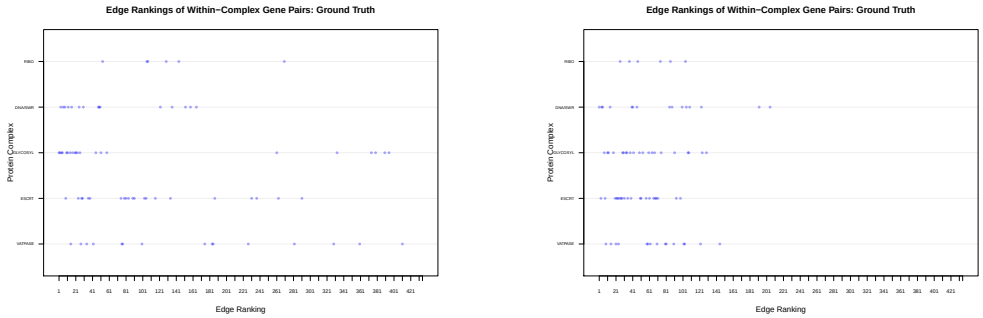

Figure 4.1: Dotplots of the ranks of within-edges for a set of 30 genes. (a) for a sample from real Schluter data (b) for a simulated edge-ranking.

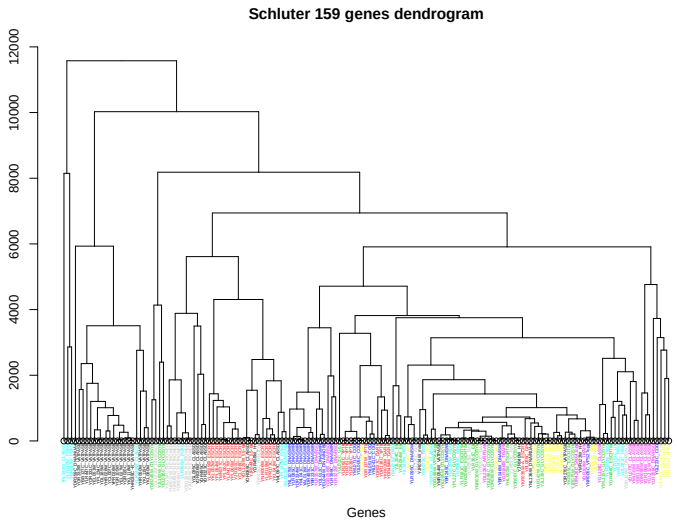

Figure 4.2: Average-linkage dendrogram of the 159 annotated Schluter genes, using Euclidean Distance. Warning: we could not get enough colors to represent each complex uniquely, but complex annotation appears after each gene name.

The Kiemer data contains 185 edges. Relative to the annotated ground truth in the Schluter dataset, 133 of these are within-edges, and 52 are between-edges.

4.2 Evaluating block structures (measures of agreement)

In all empirical studies, we wish to compare estimated or predicted block structures with the underlying truth.

Since block structures are *equivalence classes* of labelings, labels have no meaning outside of the block structure itself, and any method that compares block structures should be forbidden from comparing labels across block structures.

For this reason, it is typical to use edge-wise measures of block structure agreement. Two block structures are said to agree about an edge if they agree about its type (within or between).

4.2.1 Rand index

The **Rand index** (RI) between two block structures is the number of edges in agreement divided by the total number of edges. An edge in agreement is an edge for which both structures say "within" or both say "between".

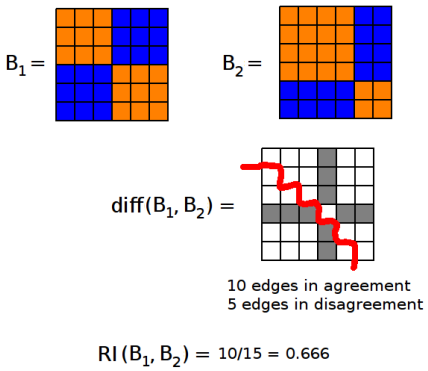

Figure 4.3: The Rand index is the proportion of edges about which the two structures agree.

A Rand index of 1 indicates complete agreement between two structures. However, in almost all cases, a Rand index of 0 is impossible to achieve.

Let:

- a: number of edges that are within according to B_1, within according to B_2
- b: number of edges that are within according to B_1, between according to B_2

- c: number of edges that are between according to B_1, within according to B_2
- d: number of edges that are between according to B_1, between according to B_2

Then $RI(B_1, B_2) = \frac{a+d}{a+b+c+d} = \frac{a+b}{N}$

4.2.2 Adjusted Rand index

One problem with the Rand index, however, is that it is not immediately clear how meaningful it is to get a specific RI value, e.g. in some cases, an RI of 0.6 can be very significant, while in others, it can easily be obtained by random guessing.

For a given block structure B_1, one can sample from its null distribution of the Rand index by sampling another block structure B_2 uniformly from the set of block structures, and computing $RI(B_1, B_2)$.

The Hubert-Arabie Adjusted Rand Index (ARI; Hubert and Arabie, 1985 [3]) is a transformation of RI, so that random block structures have an ARI of 0, and perfect matches have an ARI of 1 (see Steinley 2004 [7]).

If we let E be the expected value of the null distribution, then the ARI is a linear transformation of the RI:

$ARI_{HA}(B_1, B_2) = \frac{(a+d)-E}{N-E}$

Plugging in the formula for the expectation E, one obtains:

$ARI_{HA}(B_1, B_2) = \frac{N(a+d)-[(a+b)(a+c)+(c+d)(b+d)]}{N^2-[(a+b)(a+c)+(c+d)(b+d)]}$

4.3 Overview of case studies

For each case study, we will first look at the performance for the different methods, with and without using network data. This is in the form of an ARI boxplot. Then, we separately investigate their performance in the "with network" and "no network" cases.

We want to see if good performance correlates with high likelihood, which means that things are going well; or if they opposite is true, which suggests that we have problems.

As we will see, a problem will appear, and to better understand it, we looked at within-edge counts (which correspond to clumpiness), in order to get a better

understanding of the specific way in which our methods are proposing poor structures.

Besides the Schluter dataset, we also evaluated the performance of our methods on simulated datasets. This way, we can investigate whether any phenomena seen in the Schluter data can be explained as being due to lack of fit to the SOBA model. Details will come soon.

In the simulations, we use a single ground truth corresponding to a single resampled dataset, and simulate data from it 100 times. The following table shows the studies we perform.

n	inf. k	true k	num resampled datasets	num simulations	FlatCut	TCG	MAP
30	5	5	100	100	yes	yes	yes
60	9	9	100	100	yes	yes	no
105	12	12	1	–	yes	yes	no
117	15	15	1	–	yes	yes	no
135	20	21	1	–	yes	yes	no
145	20	26	1	–	yes	yes	no
159	20	40	1	–	yes	yes	no

Table 4.1: All experiments performed.

The study with $n = 159$ is the most inclusive, but it includes blocks of size 1, which is undesirable because biologists are not likely to target such a set, since they know that it makes the problem of genetic module identification much harder.

By excluding the 14 singleton genes, we end up with 145 genes. By excluding the 5 pairs, we end up with 135 genes. Excluding the 6 triplets, we end up with 117 genes, and excluding the 3 sets of 4, we end up with only 105.

For studies where $n > 30$, we don't run MAP, as it is too slow.

4.3.1 Data source – sampling strategy

In order to investigate how well an inference method works, it is ideal to evaluate it on a large number of datasets; a single score does not say anything about the variability in performance. In the absence of, and in addition to using many datasets, it is standard to produce a large number of artificial datasets by resampling from the real data. This way, one can capture characteristics of the original data, while

still capturing some of the variability. (In supervised learning, this usually takes the form of cross-validation).

In our case, sampling genes randomly would typically produce a variable number of complexes, and many complexes containing just 1 or 2 genes.

To approximate the setting in which genes are selected in our eventual biological application, we sample genes from the Schluter data, from a number of pre-specified protein complexes. This is done in such a way as to control the number of protein complexes in the ground truth of the resampled dataset (noted here as "true k").

4.3.2 Methods

We run the SOBA software using the following set of settings (point estimates):

- FlatCut
- TCG without network data
- TCG with network data
- MAP without network data
- MAP with network data

"FlatCut" refers to `FlatCut`, which ignores the network data. "TCG" refers to `DendroTopCutGreedy`, which prunes the dendrogram using our log-likelihood. "MAP" corresponds to `MAP-FlatCut`, optimization initialized from a flat cut.

(In the future, we may consider doing `MAP-TopCutGreedy`)

We look at the following properties of our point estimates:

- log-likelihood
- ARI with respect to the truth
- numWithinEdges – this is a measure of a block structure's clumpiness. The more unbalanced the block structure, the more within-edges it has. The "difference in number of within edges" of a point estimate means the number of within-edges in the estimated structure minus the number of within-edges of

the true structure (i.e. according to ground truth protein complex membership annotations).

In all studies below, unless stated otherwise, we used the parameter values $r = 15$, $\gamma = 0.15$, $\delta = 0.6$, and the dendrogram was generated from average-linkage clustering.

4.4 Case studies for n=30, k=5 – simulated data

For the simulated studies, we simulated 100 datasets based on a ground truth corresponding to the ground truth of one of the datasets produced by resampling, namely (`dataset061`), which has a ground truth with block sizes [6,7,7,6,4].

In Fig. 4.4, we see that, as expected, MAP performs best. Somewhat surprisingly, TCG without network data performs worse than FlatCut. And TCG *with* network data doesn't do significantly better either.

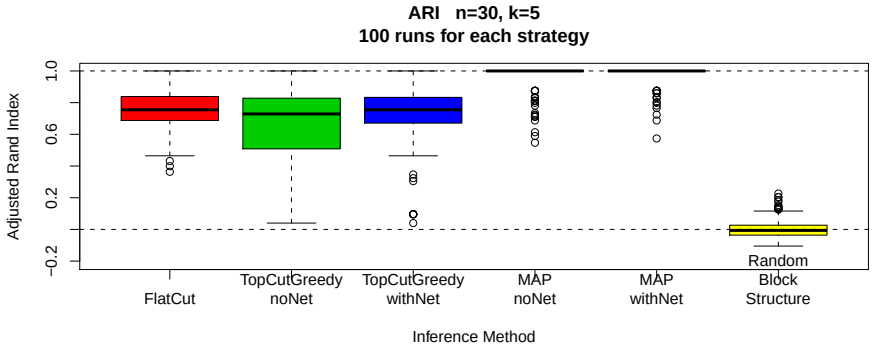

Figure 4.4: ARI for simulated data.

4.4.1 Simulated data – no network

As we see in Fig. 4.5, MAP gets the highest log-likelihoods, which is not very surprising. However, TCG, despite looking through many more structures, gets worse log-likelihoods than FlatCut. However, this pattern agrees with the ARI plot: TCG does worst in both ARI and likelihood. It is interesting to speculate how the properties of FlatCut, its simplicity or possible tendency to produce blocks of

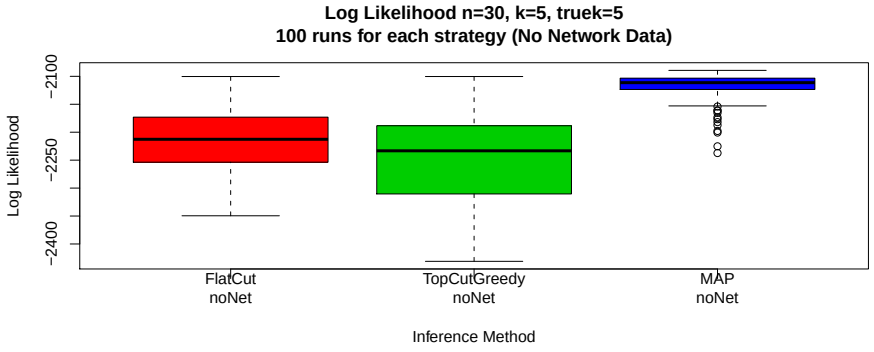

Figure 4.5: Log-likelihood, for simulated data, no network.

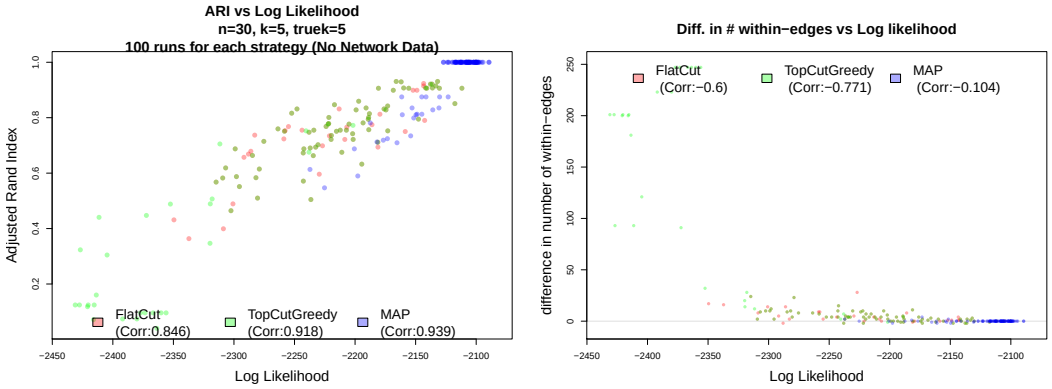

Figure 4.6: Plots for simulated data, no network (a) log-likelihood vs ARI: there is a strong correlation between log-likelihood and ARI (within each method, at least), and we see that MAP does better than FlatCut, which does better than TCG.
(b) as the log-likelihood increases, all point estimates seem to converge on the true number of within-edges.

similar sizes, may give it a better log-likelihood than TCG. Fig. 4.6(a) reassures us that higher-likelihood runs and methods correspond to higher ARIs. Likewise, Fig. 4.6(b) shows that, as the likelihood increases, the number of within-edges in the inferred structure converges on the number of within-edges in the truth. For FlatCut and TCG, there seems to be a very slight bias towards overestimating the clumpiness of the structure.

4.4.2 Simulated data – with network

As seen in Figs. 4.7 and 4.8, adding network data seems to change little. We mostly notice that likelihood-based methods have improved their performance relative to FlatCut, both in terms of ARI and log-likelihood. This is not surprising.

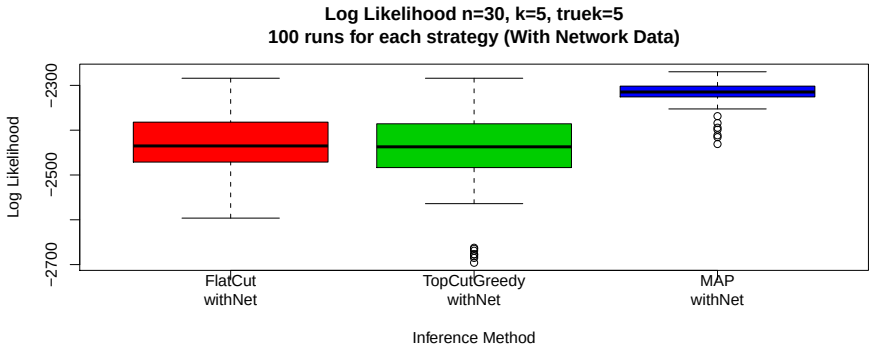

Figure 4.7: Log-likelihood, for simulated data, with network.

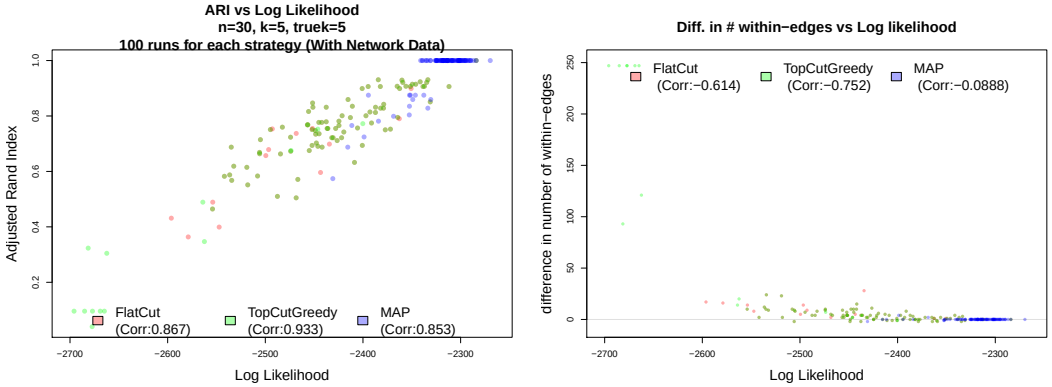

Figure 4.8: Plots for simulated data with network: (a) log-likelihood vs ARI; (b) log-likelihood vs numEdges(SOBA)-numEdges(truth)

4.5 Case studies for n=30, k=5 – real data

For the real-data studies, we produced 100 datasets using the resampling described above.

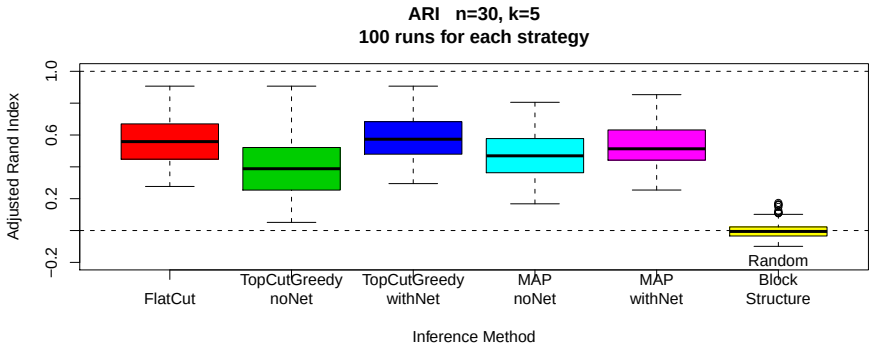

Figure 4.9: ARI for real data – Surprisingly, MAP is now performing worse than FlatCut!

In Fig. 4.9, we see that MAP is not achieving the performing baseline of FlatCut, even when it uses network data. This is worrisome.

4.5.1 Real data – no network

In Fig. 4.10, we see that MAP is achieving the highest log-likelihoods, out of the 3 methods, just as in the case of simulated data.

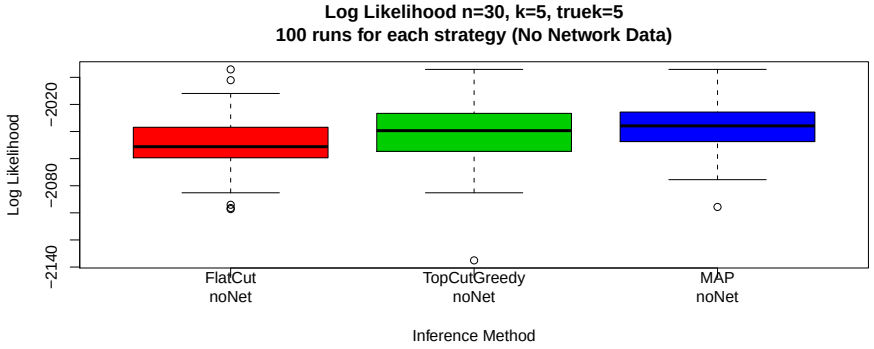

Figure 4.10: Log-likelihood for real data, no network.

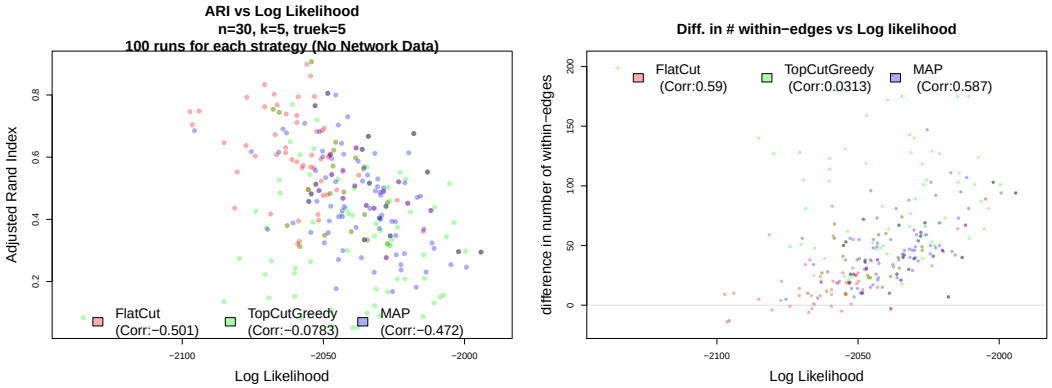

Figure 4.11: Plots for real data, no network – (a) log-likelihood vs ARI; (b) log-likelihood vs numEdges(SOBA)-numEdges(truth)

In Fig. 4.11(a), see that there is a negative relationship between log-likelihood and ARI, for each of the 3 methods, as well as for the picture as a whole! This is very surprising, if the data indeed comes from the SOBA model. This did not occur with simulated data. However, Brumm (2008) [1] anecdotally reports this effect. In Fig. 4.11(b), we see that the higher the likelihood, the greater the bias towards clumpy structures! This also did not occur with simulated data.

In particular, we observe that FlatCut has the best ARI, even though it has the lowest log-likelihoods of our methods.

The main surprise here is that better search algorithms, which return higher-scoring structures, can systematically produce worse performance, as measured by ARI! In the next chapter, we will discuss this in more detail.

4.5.2 Real data – with network

Again, MAP wins in terms of likelihood, as seen in Fig. 4.12; and, this time, log-likelihood is positively correlated with performance, as seen in Fig. 4.13(a) and the number of within-edges moves in the right direction, as seen in Fig. 4.13(b).

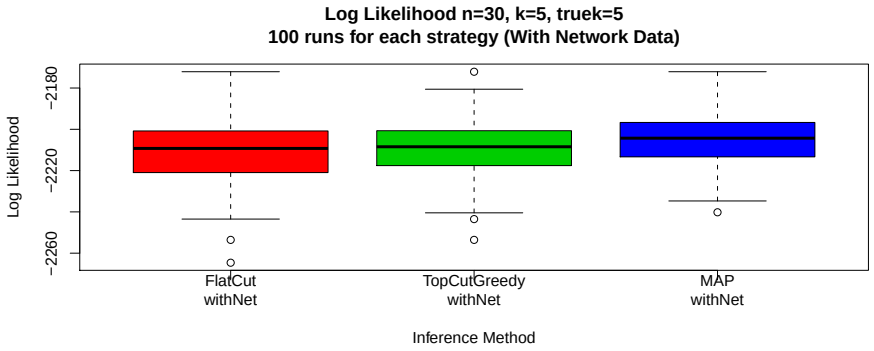

Figure 4.12: Log-likelihood, for real data, with network – Very similar to the no network case, except that, as expected, the likelihood-based methods have improved their performance.

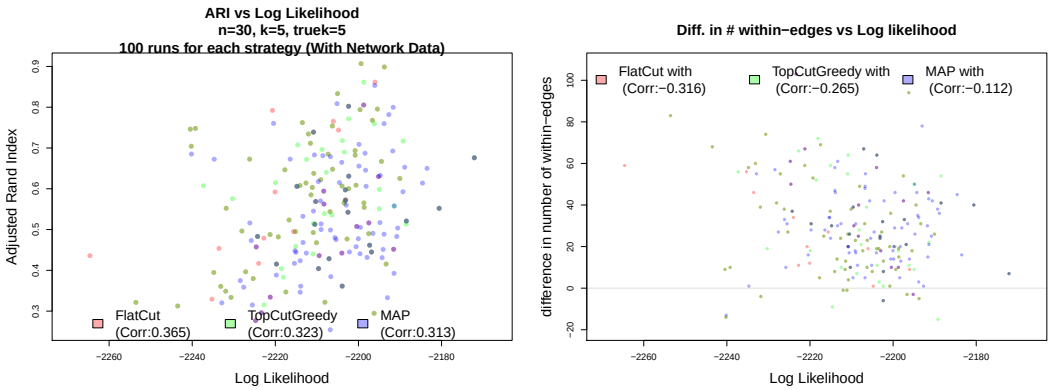

Figure 4.13: Plots for real data, with network - (a) log-likelihood vs ARI – Adding the network data seems to make the correlation positive; (b) log-likelihood vs numEdges(SOBA)-numEdges(truth) – Adding the network data cancels out the association between high log-likelihood and bias towards clumpy structures.

Again, as with simulated data, network data improves the performance of

likelihood-based methods.

The studies for $n = 60$, $k = 9$ seem to give us identical conclusions. For this reason, they are only reported in the Appendix.

4.6 Case studies for big n – real data

In these studies, we did not try to capture more than 20 blocks at a time.

The results are reported in Table 4.2, but for consistency we also produced the same ARI plots as used in the small n studies (Fig. 4.14).

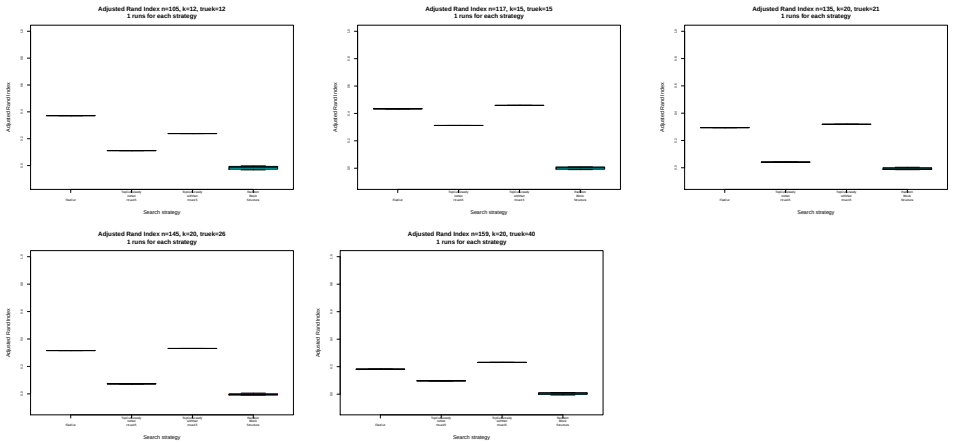

Figure 4.14: ARI for inference methods, on the big datasets – from left to right: FlatCut, TCG without network, TCG with network, random guess.

n	inference k	true k	ARI FlatCut	ARI TCG noNet	ARI TCG withNet
105	12	12	**0.3719**	0.1115	0.2384
117	15	15	0.4339	0.3122	**0.4588**
135	20	21	0.2941	0.0413	**0.3095**
145	20	26	0.3157	0.0731	**0.3310**
159	20	40	0.1818	0.0971	**0.2311**

Table 4.2: Table showing the performance of each inference method, for the big datasets.

In Table 4.2, we see the performance of each inference method, as measured by ARI. For each dataset, the winner is shown in bold.

The first thing we notice is that it becomes harder to achieve a good ARI as n gets big. This could merely reflect the increasing mismatch between the true k and the inference k; but one should also remember that in these datasets, the bigger the

n, the smaller the blocks included. It is intuitively clear that identifying structures containing small blocks is simply harder: smaller blocks have fewer within-edges, which makes it easier for them to get lost in the statistical noise.

We should mention, however, that in some settings, large n means that most blocks will increase in size, making the prediction problem *easier*.

The second thing to notice is that the relative performance of "TCG with network data" improves: that is, as n gets large, and as the dataset contains smaller and smaller complexes, TCG begins to outperform FlatCut. This could mean that TCG is better than FlatCut at recovering small blocks.

Chapter 5

Discussion and future work

5.1 Conclusion

As with much or most research in Machine Learning and Applied Statistics, our ultimate goal is to produce new combinations of models and computational infrastructures that are effective computationally as well as statistically, i.e. to improve our ability to make correct predictions effectively.

Although the evidence for our empirical advances is still rather weak, the ideas presented in this thesis seem to hold the promise of improving the state-of-art in the task of identifying genetic modules. At the moment, there are a few issues that explain our lackluster results:

- our parameter values were chosen ad-hoc, whereas they could have been elicited from a biologist, or tuned using a procedure such as an alternating maximization algorithm. The result of this might tell us to weigh the network data more or less heavily, relative to the edge-ranking data.
- the Schluter data departs significantly from the SOBA model, and in a relevant way. In particular, the maximum-likelihood estimator seems to have a strong bias towards selecting clumpy block structures, which did not occur with data simulated from the SOBA model.

Because of this bias, Brumm (2008) [1] remarks that the edge-ranking likelihood needs an adjustment, and provided one in an ad-hoc fashion. However, in

Lacerda et al (2009; technical report; make it an Appendix?), we showed analytically and empirically that this adjustment is incorrect, in the sense that maximizing Brumm's adjusted likelihood always produces the same degenerate answer, regardless of the data.

When seeing phenomena like the clumpiness bias, it may be natural to think about overfitting, and its standard solution, regularization penalties. However, in the context of block structures (or any unsupervised learning model), it is difficult to imagine how overfitting might happen. I personally take the principled stance that the best way to address the lack of fit and resulting poor performance, is to modify or generalize the probabilistic model in a way that is still fully interpretable, rather than introducing penalties that are not; and that, in long run, this will pay off in terms of prediction performance.

There is also the rather curious phenomenon of flat cuts performing so well in general, and in particular for smaller values of n. One possible explanation is that flat cuts of dendrograms produced by average-linkage clustering tend to produce evenly-sized block structures, and that this is particularly the case for our smaller datasets.

The remainder of this chapter introduces ideas for improving performance, and discusses various related topics.

5.2 Estimators: biased and unbiased

We can think of estimator bias in terms of blurring. If we start with our true structure (denoted by a small black circle in the center), we can depict statistical noise as producing a blur around it. The shades of grey show the distribution of the estimated structure (dark = high density).

The point here is that the bias of an estimator depends on the ground truth. A ground truth of [6,6,6,6,6] has as few within-edges as possible, out of structures with $n = 30$, $k = 5$. This means that any estimator will be biased in the direction of overestimating its clumpiness.

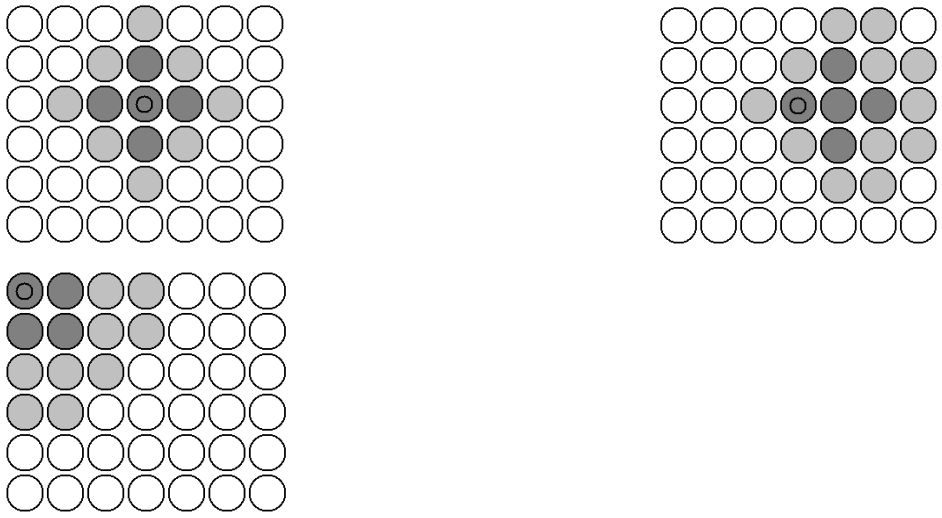

Figure 5.1: Cartoon about bias. If inference is unbiased, this distribution will be centered around the truth, as in (a). In (b), we see a bias towards more within-edges (assuming that structures to the right have more within-edges). In (c), the true structure has evenly-sized blocks: since it is impossible to get fewer within-edges, then with respect to this ground truth, inference must be biased towards clumpy structures

5.2.1 Is the likelihood biased?

Large-sample results tell us that, when the data is generated from our model, the MLE is a consistent estimator. However, when working with finite data (in our case, a single ranking), it is possible that, for some typical set of structures, there is a bias towards clumpiness. Our results with the Schluter data would suggest this, except that they are not generated from our model. Since we use a uniform prior, the MAP is identical to the MLE. Our plots of the log-likelihood vs clumpiness do not indicate that the MAP is biased, but more systematic studies could be used to address this question more formally.

However, as we saw above, any statement about bias must be qualified with a notion of what is a typical ground truth.

5.3 Is TopCutGreedy flawed?

In the previous chapter, we saw that "TCG without network data" obtained worse log-likehood than "FlatCut". This is surprising because FlatCut doesn't use the likelihood, whereas TCG selects the highest likelihood structure out of a good-sized set.

Clearly, a flat cut is a special case of a top cut, so we can safely blame this on TCG's greediness. One possible way to address this is to design a top cut search that looks ahead a number of levels before committing to splitting a block.

Also, FlatCut uses one thing that TCG does not: the order of the internal nodes. We could encourage our new search to select splits that keep the tree more balanced (either in the sense of balanced block sizes, or of removing nodes that are high up in the dendrogram). FlatCut is simply the top cut in which the set of removed nodes is as high up as possible.

5.4 Why rank, and what to rank

This project inherited the idea of ranking distances between gene profiles, but ranking, despite simplifying some aspects of the prediction problem, is not an essential component. Ranking is a lossy process, and thus it is plausible that modeling real numbers directly would give a performance boost, but that would imply very different models than what we have used.

This thesis used ranked Euclidean distance in all real-data experiments. However, it may be possible to improve prediction performance by replacing Euclidean distance with other metrics and dissimilarities, and it may be worthwhile to explore this space.

5.5 Continuous inference: inferring parameters r, γ, δ

As discussed above, it is important to use the right parameter values in our inference experiments.

A "full Bayesian" would have a prior distribution over all parameters, and update them after looking at the data. However, a more practical alternative is to find point estimates of the parameters. One possible way to do this is via an iterative

optimization algorithm: optimize B given r; then optimize r given B; and so on. Typically, this kind of procedure halts once a desired level of precision ε has been reached, but since in our case, one of the variables is discrete (namely B), we could halt when the optimization of B given r (equivalent to MAP search) reveals the same block structure as its previous iteration.

5.6 Relaxing and inferring the number of blocks k

Throughout this thesis, we worked with the setting in which k is given (if sometimes misspecified, as in the "big n" studies). This has the advantage of reducing the search space, and can help prediction in cases when the biologist already knows this number, and is only interested in the specific assignments of genes to modules.

However, it is easy to imagine a situation in which the number of blocks is unknown, and the biologist either wants to know the number of modules present, or the gene assignments, or both.

For methods like MAP and MOSS, this implies a trivial modification to our definition of neighborhood, in which we no longer exclude neighbors that have a different number of blocks. Since there is a tendency for most neighbors to have a bigger k than the present structure (unless all available labels are already assigned to a gene), we might have difficulty finding block structures with a small number of blocks, and if this is the case, it may be useful to try to control this by an auxiliary parameter controlling the number of labels used in the current iteration (i.e. the number of blocks).

TopCutGreedy already searches through structures with fewer blocks than k, and it would simply maximize the likelihood over all structures seen. We could also let it run for a few levels deeper (until the maximum value of k under consideration). We expect that TCG would benefit the most from such a change, out of all our methods.

5.7 Extending SOBA with a "junk" block

A simple conceptual extension of the SOBA model is to consider one of the blocks as the "junk block", for whom the within-edges behave like between-edges, i.e. have a strength of 1, whereas the remaining blocks have within-edges with a strength

of r.

This is desirable for biological applications in which some genes may not belong to any complex, or several genes form singleton complexes.

Now, it turns out that the probabilistic model implied by this "extension" of SOBA is in fact the same as the original SOBA model. This is evident by translating the junk block into a set of singleton blocks: all the edge strengths are preserved in this translation.

However, when using SOBA with a fixed value of k, the "junk block" would have its size limited by k. Inference in the junk-block version of SOBA is the same as running SOBA without fixing k. One trick that may be useful in such a setting is to have a special symbol to denote "junk block". This would help keep the neighborhood smaller, and allow us to express our desired constraints.

What is perhaps desired, however, is an inference method that is restricted to k **non-singleton blocks** but an arbitrary number of singleton blocks (i.e. a junk block of arbitrary size). None of the above implies a change to the probabilistic model, but only to the neighborhood relation, which is our mechanism for automatically ruling out structures that violate our constraints.

5.8 Advanced topics on posterior approximations

5.8.1 Query-specific searches, and ideas for cheaper optimization

When desiring to explore only a small set of genes (henceforth "variable genes"), MOSS for large n and k may become feasible again. This is accomplished by passing a different neighborhood relation as input to MOSS, in which neighbors modifying fixed genes are no longer considered neighbors. This guarantees that structures violating the constraint ("fixed genes shall not be reassigned") are inaccessible.

A related idea is to run MOSS (or MAP) with a small number of variable genes, periodically freeze them, and choose a different set to be variable. Yet another idea is to run MOSS or MAP on several overlapping sets of genes, and try to integrate the resulting block structures. The theory of combinatorial designs may be useful for selecting optimal sets of gene sets.

5.8.2 Consensus structures

Once one has a posterior approximation (set of block structures with estimated posterior probabilities), one can consider many notions of "consensus structure", i.e. point estimates.

- MAP: the structure with the highest posterior probability
- minAveRD: the structure in our set with the minimum Average Rand Distance from posterior approximation
- the structure corresponding to the min-cut of the graph whose weights are the colabeling probabilities
- the structure that implies a colabeling probabilities matrix that has the minimum squared distance from the colabeling probabilities matrix computed from the posterior
- the structure obtained by thresholding the colabeling probabilities and taking connected components (as in Brumm 2008 [1])

One can determine empirically which consensus works best. Preliminary studies suggest that MAP and minAveRD work best, and about as well as each other (although they disagree on a non-trivial fraction of the datasets).

5.9 Model-checking, i.e. tests for misspecification

One obvious next step in dealing with departure from our model is to create hypothesis tests reflecting possible ways in which our models are an oversimplification of the real data.

- The dotplots suggest that r is different for different blocks. To test whether two blocks have the same "r", one could take the union of these two sets of within-edges, and use a Mann-Whitney U test. To test whether all blocks have the same "r", one could do a series of Mann-Whitney U tests; or, better, a single multiple-sample version of this test.

- The dotplots also suggest that for some complexes, the within-edges do not follow our model: it looks as if within-edges from the same complex form real clusters in different parts of rank space. A statistic measuring degree of clustering should address this. Its null distribution could be computed from simulation, using the most charitable value of r (which might involve an optimization problem).

5.10 Dendrogram imperfections

The performance of inference algorithms that are based on dendrogram-pruning is bounded above by the performance on the best prune. The point is that the optimal structure will in general not be a dendrogram cut. When using methods that are restricted to the space of dendrogram cuts, it would be useful to be able to compute this upper-bound from the ground truth. This could be an interesting algorithmic problem.

5.11 Multiple edge rankings and multiple networks

It is easy to extend our models to integrate multiple edge rankings and multiple networks, by assuming that they are all independent conditional on B. To use plate notation for graphical models, this can be drawn by adding plates around the π and/or A nodes as appropriate (respectively turning them into π_i and/or A_i). Whether r,γ,δ should be included in the plate depends on whether each dataset has its own value of the parameter, e.g. if every edge-ranking π_i uses a different r, then the plate should engulf r into r_i.

5.12 Priors on block structures

Since the number of possible block structures for any given finite set of nodes is finite, it is consistent to use a uniform prior, as was done throughout this thesis.

A Bayesian, however, may wish to incorporate biologists' prior knowledge about e.g. suspected number of blocks, typical granularities, or suspected associations between pairs of genes; but regardless of what we believe is closest to the biological truth, it could still be more useful to prefer certain types of structure,

e.g. certain levels of granularity, and thus use an “instrumental” prior accordingly.

Bibliography

[1] J. Brumm, E. Conibear, W. W. Wasserman, and J. Bryan. Discovery and expansion of gene modules by seeking isolated groups in a random graph process. PhD Thesis, UBC, 2008. → pages ii, 3, 6, 10, 11, 38, 43, 49

[2] F. Crick. On protein synthesis. pages 139–163. Symp. Soc. Exp. Biol. XII, 139-163., 1958. → pages 1

[3] Hubert and Arabie. Comparing partitions. pages 193–218. Journal of Classification, 1985. → pages 31

[4] L. Kiemer, S. Costa, M. Ueffing, and G. Cesareni. WI-PHI: a weighted yeast interactome enriched for direct physical interactions. pages 932–43. Proteomics 2007, Mar; 7(6) – PMID: 17285561, 2007. → pages 27, 28

[5] R. Neal. Probabilistic inference using Markov Chain Monte Carlo methods. Technical Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto, 1993. → pages 21

[6] C. Schluter, K. K. Lam, J. Brumm, B. W. Wu, M. Saunders, T. H. Stevens, J. Bryan, and E. Conibear. Global analysis of yeast endosomal transport identifies the vps55/68 sorting complex. pages 1282–94. Mol Biol Cell. 2008 Apr;19(4) – PMID: 18216282., 2008. → pages 27, 28

[7] Steinley. Properties of the Hubert-Arabie Adjusted Rand Index. pages 386–96. Psychol. Methods, 2004. → pages 31

Appendix A

Appendix – Guide to the SOBA code

A.1 Dependencies

The SOBA code is implemented in R, and uses the following packages:

- Hash 1.X

A.2 Organization

We organize our studies as follows: datasets (whether real or simulated) are folders (directories) that contain files named `*edgeRanking*` and `*network_data*` and a number of run folders, which match the pattern `run*` or `done*` (each folder gets renamed from "run*" to "done*" once it has been run). Each run folder has its own setting, in a file named `config.csv`, which specifies inference options and parameters.

collection folder $>$ dataset folders $>$ run folders

A.3 Configuration files

All configuration files are in comma-separated format.

A.3.1 config.csv

`config.csv` is used for inference, by `run.R`:

```
"k","r","gamma","delta","searchStrategy","hclustMethod",
"missingNetworkDataModel","zeta","eta"
5,15,0.6,0.15,"DendroFlatCut","average","ABSENT",,
```

In this example, inference assumes that $k = 5$, $r = 15$, $\gamma = 0.6$, $\delta = 0.15$, uses `DendroFlatCut` as the inference method, from an average-linkage clustering tree; and assumes that missing edges are absent (no other setting is supported).

A.3.2 configurations.csv

`configurations.csv` specifies a collection of `config.csv` files (one per line). It is used by batchMakeRuns.R, for generating a collection of run folders for each dataset, and populating each run folder with its own `config.csv`.

As an example, the following `configurations.csv` can be used to set up an experiment to investigate the effect of changing the value of r from 5 to 10:

```
"k","generateRanking","r","generateNetwork","gamma","delta",
"searchStrategy","hclustMethod","missingNetworkDataModel
","zeta","eta"
5, TRUE, 5, TRUE, 0.6, 0.15, "DendroTopCutGreedy","average","ABSENT",,
5, TRUE, 10, TRUE, 0.6, 0.15, "DendroTopCutGreedy","average","ABSENT",,
```

A.3.3 config-sim.csv

`config-sim.csv` is for generating new datasets, and is used by `simulate-multiple.R`. The ground truth can be specified in AABB format, or from a `ground_truth.txt` file:

```
"truth", "k", "generateRanking", "rtrue", "generateNetwork",
"gamma", "delta", "baseName", "nSims", "zeta", "eta"
"AAABBBBCCCCCDDDDDDDEEEEE", 5, TRUE, 15, TRUE, 0.6, 0.15, "dataset_061-sim
```

In this example, 100 datasets will be generated from the same ground truth of type [3,4,5,7,5]; these datasets will including edge ranking *and* network data, and

the folders will be named from `dataset_061-sim001` to `dataset_061-sim100`. (It is desirable to name datasets in such a way if the ground truth was inspired by the ground truth of `dataset061`)

A.4 How to produce datasets and run experiments in batch

Define a collection folder, containing multiple dataset folders. Inside the collection folder:

- (a) generate dataset folders, e.g. by (a1) using batchSimulate.R (currently simulate-multiple.R), or (a2) resample-genes.R (Timothy's code, real name is unknown). This will produce dataset folders, each containing appropriately-named data files (`*edgeRanking*` and sometimes `*network_data*`)
- (b) Rscript batchMakeRuns.R to generate run folders. For each run folder, this script copies the data files from the dataset folder, and produces a `config.csv`.
- (c) Rscript batchRun.R from the dataset folder (currently batchSobaRun.R). This produces outputs inside each run folder, and renames them from "run" to "done".
- (d) Rscript batchEvaluate.R (currently batchTimothyEvaluation.R). This produces a `SOBAPerformanceEvaluation` folder inside each run folder.
- (e) Rscript batchVisualize.R (currently visualizeRIForBatchRun.R), which produces plots in the collection folder.