# Learning Gene Interactions and Networks from Perturbation Screens and Expression Data

Kieran Elmes

Supervised by Alex Gavryushkin, Zhiyi Huang, and Matthieu Vignes

a thesis submitted for the degree of

Master of Science

at the University of Otago, Dunedin, New Zealand.

August 31, 2020

# Contents

## Abstract

We investigate a variety of methods to first discover and then understand genetic interactions. Beginning with pairwise interactions, we propose a method for inferring pairwise gene interactions en masse from short-interfering RNA screens. We use the siRNA off-target effects to form a matrix of knocked-down genes, and consider the observed fitness to be a linear combination of individual and pairwise effects in this matrix. These effects can then be inferred using a variety of statistical learning methods. We evaluate two such methods for this task, `xyz` and `glinternet`. Using either method, we are able to find interactions in small simulated data sets. Neither method scales to genome-scale data sets, however. In our larger simulations both methods suffer from scalability problems, either with their accuracy or running time.

We overcome these limitations by developing our own lasso-based regression method, which takes into account the binary nature of our perturbation screens. Using a compressed sparse representation of the pairwise interaction matrix, and parallelising updates, we are able to run this method on exome-scale data. Generalising from pairwise interactions we then consider network models, in which pairwise gene interactions form edges of a graph. Such networks are often understood in terms of functional modules, groups of genes that act together to perform a task. We develop a method that combines pairwise interaction and gene expression data to effectively find functional modules in simulated data.

## Introduction

Genetic interactions, in which the cumulative effect of genes differs from the sum of their individual effects, can be modelled in a variety of ways, including as pairwise epistatic effects [15], and interaction networks [84]. We investigate a variety of methods to first discover and then understand such interactions. Starting with the simplest approach, we assume that fitness can be modelled as a linear combination of the effects genes and their combinations. We further assume that individual and pairwise effects are the most significant, and leave out higher order effects. In Chapter 1, we propose a method for inferring pairwise gene-interactions en masse from short-interfering RNA screens by exploiting their many off-target effects using this model. These effects can then be inferred using a variety of statistical learning methods, and we evaluate two such methods for this task, `xyz` [89] and `glinternet` [53]. Using data simulations based on real siRNA libraries, we evaluate both methods on data sets ranging from 100 to $1,000$ genes, and conclude that the general approach works well at a small scale, particularly for `glinternet`. We aim to use this approach on human genome-scale data sets, however, and in our larger simulations both methods suffer from scalability problems with either their accuracy or running time.

We overcome these limitations in Chapter 2 by developing our own lasso-based regression method, which takes into account the binary nature of our perturbation screens. We begin with an existing fast approach to lasso regression based on cyclic coordinate descent, and investigate a number of potential improvements. Since the main barrier to parallelisation is overlap in columns of the input matrix, we first consider parallel parameter updates where columns do not overlap, then where columns only overlap to a small degree. Finally we update parameters at random, finding that this outperforms both earlier methods. To account for the sparse binary nature of our data, we store columns of the matrix as a list of entry positions. These offsets are delta encoded, and then compressed. After comparing a number of state of the art integer compression methods, we use Simple-8b [1], concluding that it provides the best balance between hardware requirements and performance. We are then able to run this method on genome-scale data.

Generalising from pairwise interactions, we then consider network models in which pairwise gene interactions form edges of a graph. Such networks are often understood in terms of functional modules, groups of genes that act together to perform a task, and we focus on extracting these functional modules via graph clustering. In Chapter 3 we develop a method to do so, combining pairwise networks with gene expression data. The gene expression data is first converted into a graph of correlations, with small correlations filtered out. Both the gene expression and pairwise interaction graphs are then simultaneously clustered with the SLPA algorithm. Finally, only proposed clusters with a sufficiently high modularity density score are kept. We find this combination produces more accurate results than clustering either data set alone, and our approach is able to effectively find functional modules in simulated data.

Chapter 1 is a modified form of the work in Elmes et al. [21], co-authored by Fabian Schmich, Ewa Szczurek, Jeremy Jenkins, Niko Beerenwinkel, and Alex Gavryushkin. The research plan for the use of `xyz`, analysis of real and larger data, and scalability was conceived collaboratively, and carried out by the author. Discussion is primarily the author's own. The siRNA matrix simulations, use of `glinternet`, and analysis of small data in this chapter were initially developed by co-authors, and subsequently modified by the author. Chapters 2 and 3 are entirely the author's own.

# Chapter 1

# Learning Epistatic Gene Interactions from Perturbation Screens

Co-authored By: Fabian Schmich, Ewa Szczurek, Jeremy Jenkins, Niko Beerenwinkel, and Alex Gavryushkin

## 1.1 Introduction

Genetic interactions are also referred to as epistasis, a term that originates from the field of statistical genetics and describes genetic contributions to the phenotype that are not linear in the effects of single genes [98, 16]. Considering two genes at a time, positive and negative epistasis refer to a greater and smaller effect, respectively, of the double mutant genotype than expected from the two single mutant genotypes relative to the wild type. In genetics, the phenotype of primary interest is the reproductive success of a cell, which is commonly termed fitness [65]. In this context, a fitness landscape is the mapping of each combination of possible configurations of gene mutations to a fitness phenotype [18].

The knowledge of fitness landscapes is highly relevant for personalized disease treatment [42]. In cancer, for example, genetic aberrations result in cells with increased somatic fitness, for instance, by evading apoptosis or gaining the ability to metastasise. This increase subsequently promotes post-metastatic tumour development [29]. A major challenge in cancer therapy is the fact that many genes with driving mutations cannot be adequately targeted for inhibition due to toxic side effects and rapid development of drug resistance [22, 32]. To overcome this challenge, a strategy based on the inhibition of genes that interact with genes with cancer driving alterations was proposed [3]. This strategy is based on the principle of synthetic lethality [42, 38, 64], the extreme case of negative epistasis, where single mutants are compatible with cell viability but the double mutant results in cell death. Identifying synthetic lethal gene interactions allows targeting cancer cells in which one of the two genes is mutated, by using drugs that affect the other. In the presence of this drug, the cancer cell lineage

will no longer be viable [10].
The identification of fitness landscapes is however a very challenging task, simply due to the exponential growth of the space of interactions. For yeast, for example, it has been shown to be feasible to experimentally perform 75% of all pairwise knockouts [17]. However, in humans, with approximately 20,000 protein-coding genes, this would constitute to almost 200 million experiments to test all pairwise interactions. An approach that has been successfully applied to identify synthetic lethality *in vitro* is large-scale perturbation screening of human cancer cell lines using RNA interference [83, 48, 58, 19]. However, this strategy only allows cataloguing synthetic lethal gene pairs where one gene is always specific to the screened cell line. While these methods may be sufficient for the identification of a few promising targets for cancer therapy, they do not allow us to estimate general pairwise gene interactions at the human exome scale.
Short-interfering RNAs (siRNAs), the reagents used in RNAi perturbation screening, exhibit strong off-target effects, which results in high numbers of false positives rendering the perturbations hard to interpret [35]. While this is usually conceived as a problem, here we take advantage of this property for the estimation of genetic interactions [78, 82, 91]. We propose a novel approach for the second order approximation of a human fitness landscape by inferring the fitness of single gene perturbations and their pairwise interactions from RNAi screening data (Figure 1.1). Our approach is not restricted to interactions with mutant genes of a specific cell line or explicit double knockdowns. We leverage the combinatorial nature of sequence-dependent off-target effects of siRNAs, where each siRNA in addition to the intended on-target knocks down hundreds of additional genes simultaneously. Not distinguishing between on- and off-targeted genes, we consider each siRNA knockdown as a combinatorial knockdown of multiple genes. Hence, every large-scale RNAi screen, though unintended, contains large numbers of observations of high-order combinatorial knockdowns and provides a rich source for the extraction of pairwise epistasis. These off-target effects have previously been used to improve inference of signalling pathways among a small number (on the order of a dozen) genes [82, 91]. Here, however, we attempt to use it to discover epistatic gene pairs in a genome-wide fashion (i.e. among tens of thousands of genes).
Our approach is formulated as a regularised regression model. It can also be deployed for the estimation of epistasis from phenotypes other than fitness, such as for instance phenotypes that measure the activity of disease-relevant pathways, e.g. for pathogen entry [74], TGF$\beta$-signalling [79], or WNT-signalling [88]. Long term, the identification of disease-relevant epistatic gene pairs may allow the design or re-purposing of agents for combinatorial therapy with the potential to improve the efficacy of drugs.
In solving this model, we adapt two recent statistical learning methods, namely `glinternet` [53] and `xyz` [89] to select genes and gene-pairs with non-zero effects on fitness, and evaluate both models on simulated data from real RNAi libraries. We vary the signal-to-noise ratios, number of true gene–gene interactions, number of observations per double knockdown and effect size for epistasis. We find that, within ranges that are realistic to real RNAi data, both approaches are capable of inferring pairwise epistasis with favourable precision and sensitivity when only a small number of genes are involved in interactions. In several tests `glinternet` continued to infer correct interactions up to several thousand genes, however the run time prohibits more thorough

testing. To demonstrate the model on a real data set, we use the perturbation data from [74]. Using `glinternet`, we search for interactions between kinases, and report the most significant results.
Our simulations are performed using `R`, and the source code is available at: `https://github.com/bioDS/xyz-simulation`.

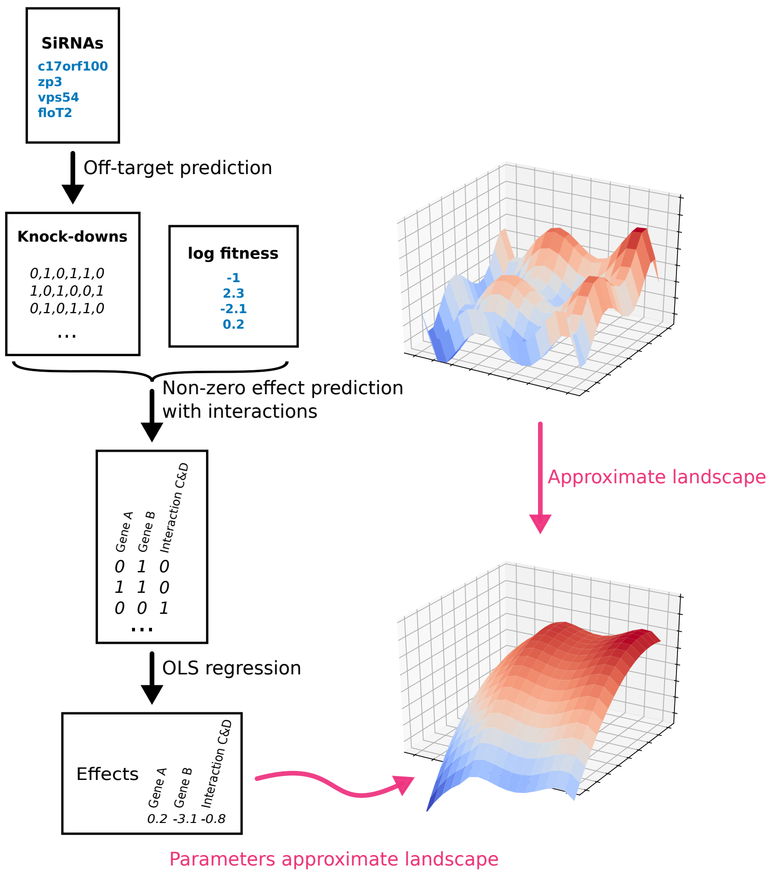


Figure 1.1: RNAi fitness landscape model. Black arrows indicate outputs that are actually produced. Red arrows indicate theoretical output.

## 1.2 Methods

We fix the binary alphabet $\Sigma = \{0, 1\}$ representing the two possible states in a perturbation experiment. The value zero denotes the normal state of the gene (unperturbed wild type), whereas the value one indicates knockdown of the gene (perturbed). For $p$ genes we denote by $\Sigma^p$ the set of binary sequences of length $p$, indicating the perturbation status of each gene. Any subset $\mathcal{P} \subseteq \Sigma^p$ is called a perturbation space and its elements are called perturbation types. If the perturbations are genetic mutations, then the perturbation types are genotypes.

### 1.2.1 Fitness landscapes and epistasis

In the following, we focus on fitness landscapes, but would like to note that the theory also holds for any mapping of perturbation type to phenotype. A fitness landscape is a mapping $f : \mathcal{P} \to \mathbb{R}_+$ from perturbation type space to non-negative fitness values. Genetic interactions are a property of the underlying fitness landscape [5]. For $p = 2$ genes, the perturbation type space $\mathcal{P} = \{0,\, 1\}^2$ contains the wild-type 00, two single perturbations 01 and 10, and the double perturbation 11. The fitness landscape $f : \{0, 1\}^2 \to \mathbb{R}_+$ can be written as

$$\begin{aligned} f(0,0) &= \beta_0 \\ f(1,0) &= \beta_0 + \beta_1 \\ f(0,1) &= \beta_0 + \beta_2 \\ f(1,1) &= \beta_0 + \beta_1 + \beta_2 + \beta_{1,2} \end{aligned}$$

for parameters $\beta_i \in \mathbb{R}$. $\beta_0$ is called the bias, $\beta_1$ and $\beta_2$ main effects, and $\beta_{1,2}$ the interaction. Epistasis is defined as

$$\varepsilon = f(0,0) + f(1,1) - f(0,1) - f(1,0) \tag{1.1}$$

It measures the deviation of the fitness of the double knockdown from the expectation under a linear fitness model in the main effects. We see that $\varepsilon = \beta_{1,2}$.

#### Fitness landscape model

It is challenging to generalise the notion of epistasis (Equation 1.1), because in higher dimensions, many more types of genetic interactions exist [5], even when restricting to pairwise interactions. In general, it will be impossible to estimate all interactions encoded in the fitness landscape reliably from data. In the following, we show how to assess marginal and conditional pairwise epistasis. For $p \geq 1$ genes, we consider the Taylor expansion of the fitness landscape

$$f(x_1,\, \dots,\, x_p) = \beta_0 + \sum_i x_i \beta_i + \sum_{i<j} x_i x_j \beta_{i,j} + \sum_{i<j<k} x_i x_j x_k \beta_{i,j,k} + \dots \tag{1.2}$$

Ignoring interactions of order 3 and higher we obtain the more computationally tractable approximation:

$$f(x_1,\, \dots,\, x_p) \approx \beta_0 + \sum_i x_i \beta_i + \sum_{i<j} x_i x_j \beta_{i,j} \tag{1.3}$$

#### Conditional epistasis

For two genes $i$ and $j$ and a fixed set of background perturbations $b = (x_1,\, \dots,\, x_{i-1},$ $x_{i+1},\, \dots,\, x_{j-1},\, x_{j+1},\, \dots, x_p) \in \mathbb{R}_+^{p-2}$ we define conditional epistasis between gene $i$ and

$j$ given $b$ as

$$\begin{aligned}\varepsilon_{i,j|b} = f(&x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_{j-1}, 0, x_{j+1}, \ldots, x_n) \\ &+ f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_{j-1}, 1, x_{j+1}, \ldots, x_p) \\ &- f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_{j-1}, 0, x_{j+1}, \ldots, x_p) \\ &- f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_{j-1}, 1, x_{j+1}, \ldots, x_p) \end{aligned} \tag{1.4}$$

*Proposition* 1. For the fitness landscape model (1.3), the interaction terms $\beta_{i,j}$ are independent of $b$ and equal to conditional epistasis, that is, $\varepsilon_{i,j|b} = \beta_{i,j}$.

*Proof.* Without loss of generality, we can consider $(i, j) = (1, 2)$. Let $b = (x_3, \ldots, x_p)$. In model (1.3) we have

$$\begin{aligned}\varepsilon_{1,2|b} &= f(0, 0, x_3, \ldots, x_p) + f(1, 1, x_3, \ldots, x_p) \\ &\quad - f(1, 0, x_3, \ldots, x_p) - f(0, 1, x_3, \ldots, x_p) \\ &= \beta_0 + \left(\beta_0 + \beta_1 + \beta_2 + \beta_{1,2} + \sum_{i>2} x_i\beta_{1,i} + \sum_{i>2} x_i\beta_{2,i}\right) \\ &\quad - \left(\beta_0 + \beta_1 + \sum_{i>2} x_i\beta_{1,i}\right) - \left(\beta_0 + \beta_2 + \sum_{i>2} x_i\beta_{2,i}\right)\end{aligned} \tag{1.5}$$

All terms except the interaction $\beta_{1,2}$ cancel out, therefore $\varepsilon_{1,2|b} = \beta_{1,2}$. □

### Marginal epistasis

The marginal fitness landscape of genes $i$ and $j$ is

$$f_{i,j}(x_i, x_j) = \sum_{\{x_k \in \{0,1\} | k \neq i,j\}} f(x_1, \ldots, x_p) \tag{1.6}$$

and marginal epistasis between genes $i$ and $j$ is the epistasis of the marginal fitness landscape,

$$\varepsilon_{i,j} = f_{i,j}(0, 0) + f_{i,j}(1, 1) - f_{i,j}(1, 0) - f_{i,j}(0, 1) \tag{1.7}$$

For example, for $p = 3$ genes, marginal epistasis between gene 1 and 2 is

$$\begin{aligned}\varepsilon_{1,2} = [f(0, 0, 0) + f(0, 0, 1)] &+ [f(1, 1, 0) + f(1, 1, 1)] \\ &- [f(1, 0, 0) + f(1, 0, 1)] - [f(0, 1, 0) + f(0, 1, 1)]\end{aligned} \tag{1.8}$$

*Corollary* 1. For the fitness landscape model (1.3), the interaction terms $\beta_{i,j}$ are related to marginal epistasis via $\varepsilon_{i,j} = 2^{p-2}\,\beta_{i,j}$.

*Proof.* From Proposition 1 we have that conditional epistasis for a pair of genes $(i, j)$ and a fixed genetic background of the remaining $p-2$ genes equals $\beta_{i,j}$. There are $2^{p-2}$

such genetic backgrounds, and the conditional epistasis is the same for all of them. □

Thus, in the fitness landscape model (1.3), which contains all main effects and pairwise interactions, but no interactions of higher order, the interaction terms $\beta_{i,j}$ alone determine conditional and marginal epistasis of the fitness landscape.

### 1.2.2 Estimation of epistasis from RNAi perturbation screens

In *in vitro* RNAi experiments cells are perturbed by reagents, such as siRNA, shRNA, and dsRNA [80], each targeting a specific gene for knockdown. In recent years, it has been shown [35] that siRNAs exhibit strong sequence-dependent off-target effects, such that, in addition to the intended target gene, hundreds of other genes are knocked down. Thus, we can regard siRNA perturbation experiments as combinatorial knockdowns affecting multiple genes simultaneously. On the basis of the fitness landscape model (1.3), we propose a regression model for the estimation of epistasis from RNAi data. This inference is only feasible because of the unintended combinatorial nature of siRNA knockdowns.

#### Perturbation type space

For an RNAi-based perturbation screen, the perturbation type space $\mathcal{P} = \{g_1, \ldots, g_n\}$ is represented as the $n \times p$ matrix $\boldsymbol{X}$ that contains $g_i$ in row $i$. Based on the nucleotide sequences of the reagents, perturbations can be predicted by models for micro RNA (miRNA) target prediction [50]. We use $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_p$ to denote the $p$ column vectors of $\boldsymbol{X}$ for genes $1, \ldots, p$ and denote by $\boldsymbol{X}_i \circ \boldsymbol{X}_j$ the column vector consisting of the element-wise products of the entries of $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$. As a measure of fitness, we use the vector $\boldsymbol{Y} \in \mathbb{R}^n_+$, denoting the number of cells present after siRNA knockdown.

#### Regression model

We aim to estimate the conditional epistasis $\beta_{i,j}$ between the $\binom{p}{2}$ pairs of genes $(i, j) \in \{1, \ldots, p\}^2$ from all combinatorial gene perturbations in the screen represented in the $n \times p$ matrix $\boldsymbol{X}$, and the $n \times 1$ vector of fitness phenotypes $\boldsymbol{Y}$. Based on (1.3) we regress phenotype Y on perturbations X,

$$\mathbb{E}\left[\boldsymbol{Y} \mid \boldsymbol{X}\right] = \beta_0 + \sum_i \boldsymbol{X}_i \beta_i + \sum_{i<j} (\boldsymbol{X}_i \circ \boldsymbol{X}_j)\beta_{i,j} \tag{1.9}$$

The estimated $\beta_{i,j}$ are interpreted as the expected change in the response variable $\boldsymbol{Y}$ per unit change in the predictor variable $(\boldsymbol{X}_i \circ \boldsymbol{X}_j)$ with all other predictors held fixed [60]. From Corollary 1 it follows that estimates for marginal epistasis $\varepsilon_{i,j}$ can be obtained by multiplication of $\beta_{i,j}$ with the constant $2^{p-2}$.

#### Inference

We aim to infer the regression parameters $\boldsymbol{\beta} = \left(\beta_0, \boldsymbol{\beta}_{\{i:i>0\}}, \boldsymbol{\beta}_{\{i,j:i<j\}}\right)$. Since it is infeasible to directly perform least squares linear regression on the matrix containing

all $\binom{p}{2}$ interactions, we use a two-stage process. First, we use either the group lasso regularisation package `glinternet` [53], or the `xyz` interaction search algorithm [89] to select non-zero interactions. This variable selection step is the main computational challenge.
When using `glinternet`, we infer parameters $\boldsymbol{\beta} = \left(\beta_0, \boldsymbol{\beta}_{\{i:i>0\}}, \boldsymbol{\beta}_{\{i,j:i<j\}}\right)$ by minimising the squared-error loss function

$$\mathcal{L}(\boldsymbol{Y}, \boldsymbol{X}; \boldsymbol{\beta}) = \frac{1}{2} \left\| \boldsymbol{Y} - \left( \beta_0 + \sum_i \boldsymbol{X}_i \beta_i + \sum_{i<j} (\boldsymbol{X}_i \circ \boldsymbol{X}_j) \beta_{i,j} \right) \right\|_2^2 \tag{1.10}$$

under the *strong hierarchy* constraint

$$\beta_{i,j} \neq 0 \Rightarrow \beta_i \neq 0 \text{ and } \beta_j \neq 0. \tag{1.11}$$

This constraint allows conditional epistasis between gene $i$ and $j$, i.e., $\beta_{i,j} \neq 0$, only if both single-gene effects $\beta_i$ and $\beta_j$ are present and constrains the search space. Lim and Hastie (2015) show that this model can be formulated as a linear regression model with overlapped group lasso (OGL) penalty [36], where, in contrast to the group lasso [102], each predictor can be present in multiple groups.
To perform the variable selection, `xyz` searches for pairs $(i, j)$ that maximise $\mathbf{Y}^T X_i X_j$. These are the interaction effects that account for the largest component of the response $Y$. While `xyz` can be used directly to find the largest interactions, we used `xyz_regression` to estimate all interactions. `xyz_regression` solves the following elastic-net problem [89]

$$\min_{(\beta_0, \beta) \in \mathbb{R}^{p+1}, \theta \in \mathbb{R}^{p(p+1)/2}} \left[ \frac{1}{2n} \sum_{i=1}^{N} (y_i - \beta_0 - X_i^T \beta - w_i^T \theta)^2 + \lambda (P_\alpha(\beta) + P_\alpha(\theta)) \right], \tag{1.12}$$

where

$$W \in \mathbb{R}^{n \times p(p+1)/2} = (X_1 \circ X_1, X_1 \circ X_2, \ldots, X_1 \circ X_p, X_2 \circ X_2, \ldots, X_p \circ X_p) \tag{1.13}$$

is the matrix of interactions, and

$$P_\alpha(\beta) = (1 - \alpha) \frac{1}{2} ||\beta||_{\ell_2}^2 + \alpha ||\beta||_{\ell_1} \tag{1.14}$$

is the elastic-net penalty.
The parameter $\alpha$ decides the compromise between the ridge-regression penalty ($\alpha = 0$) and the lasso penalty ($\alpha = 1$). We left the default value of $\alpha = 0.9$. The solution is found iteratively, with only a particular set of beta values are allowed to be non-zero at each iteration. In every iteration, the beta values that violate the Karush–Kuhn–Tucker conditions are added to this set. Rather than being computed directly, these beta values are found using the `xyz` algorithm. We followed the recommendation in [89] and used $L = \sqrt{p}$ projections to find the strong interactions. Our own tests in Appendix A.3 also suggest that further projections do not improve performance.

Second, once the non-zero effects have been estimated using either `glinternet` or `xyz`, we construct a matrix $X'$ with all elements of the set $\{\boldsymbol{X}_i | \boldsymbol{X}_i \neq \mathbf{0}\} \cup \{\boldsymbol{X}_i \circ \boldsymbol{X}_j | \boldsymbol{X}_i \cdot \boldsymbol{X}_j \neq 0\}$ as columns, in an arbitrary order. We then fit $Y \sim X'\beta$ using R's `lm` least squares linear regression to calculate the coefficient estimates and corresponding p-values. We adjust the p-value to control the false discovery rate with the method of Benjamini and Hochberg [6], and refer to this adjusted value as the q-value. Given this two-step procedure, we do not expect these values to be the same as if they were calculated using the complete interaction matrix. We are nonetheless able to distinguish between more and less significant effects, with the caveat that the $p < 0.05$ cut-off is completely arbitrary.

### 1.2.3 Software

The overlapped group lasso for strongly hierarchical interaction terms is implemented in the R-package `glinternet` 1.0.10 by Lim and Hastie [53] and available through the *Comprehensive R Archive Network* (CRAN) at `https://cran.r-project.org/web/packages/glinternet/`. The `xyz` algorithm is implemented in `xyz` 0.2 by Gian-Andrea Thanei [89] available at `https://cran.r-project.org/web/packages/xyz/`. The simulations are run using a version of this software that also contains a trivial bug fix, available at `https://github.com/bioDS/xyz-simulation`. For the data simulation, analysis and visualisation, we used the R-packages `Matrix` 1.2.6, `dplyr` 0.4.3, `tidyr` 0.4.1 and `ggplot2` 2.1.0. All simulations are performed using `R` 3.2.4.

### 1.2.4 Simulation of RNAi data

The data simulation followed a three-step procedure. First, we simulate the siRNA–gene perturbation matrix $\boldsymbol{X}$ based on real siRNA libraries. Second, main effects $\beta_i$ and conditional epistasis between pairs of genes $\beta_{i,j}$ are sampled. Based on $\boldsymbol{X}$ and $\boldsymbol{\beta}$, we then sample fitness phenotypes $\boldsymbol{Y}$ from our model (1.3) and add noise to match specific signal-to-noise ratios [31]

$$\mathrm{SNR} = \frac{\mathrm{Var}\left(\mathbb{E}\left[\boldsymbol{Y} \mid \boldsymbol{X}\right]\right)}{\mathrm{Var}\left(\boldsymbol{Y} - \mathbb{E}\left[\boldsymbol{Y} \mid \boldsymbol{X}\right]\right)}. \tag{1.15}$$

Details for each step including parameter ranges are as follows.
We simulate siRNA–gene perturbation matrices based on four commercially available genome–wide libraries for 20 822 human genes from Qiagen with an overall size of 90 000 siRNAs. First, we predict sequence dependent off-targets using TargetScan [27] for each siRNA as described in [78]. We threshold all predictions to be 1 if larger than zero and 0 otherwise. Then, we sample $n = 1\,000$ siRNAs from $\{1, \ldots, 90\,000\}$ and $p = 100$ genes from $\{1, \ldots, 20\,822\}$ without replacement and construct the $n \times p$ binary matrix $\boldsymbol{X}$. Hence, each row $i$ of $\mathbf{X}$ then contained the perturbation type $g_i = (x_{i,1}, \ldots, x_{i,p})$. We simulate $q \in \{5, 20, 50, 100\}$ non-zero conditional epistasis terms $\beta_{i,j}$ between genes $i$ and $j$ from all observed combinatorial knockdowns, i.e. if the simulated screen contained siRNAs that target both genes. This is a necessary condition for the identifiability of $\beta_{i,j}$, as otherwise, according to the model (1.9), $\beta_{i,j}$ will be multiplied by a

zero vector $X_i \circ X_j = \mathbf{0}$. The effect size of the $\beta_{i,j}$ is sampled from $\mathrm{Norm}(0, 2)$. In order to maintain a strong hierarchy, we subsequently simulate for each interaction $\beta_{i,j}$ both main effects $\beta_i$ and $\beta_j$. Further, we add $r \in \{0, 20, 50, 100\}$ additional main effects. The effect sizes of the main effects are sampled from $\mathrm{Norm}(0, 1)$, so that the variance in the response fitness phenotypes are split in a ratio of 1:2 between main effects and interactions.
In order to model synthetic lethal pairs, interactions with effect strength of $-1000$ (on log scale) are added to the simulated data. Since lethal interactions may occur with little or no main effect present [38], we allow these pairs to violate the strong hierarchy and do not add main effects. This is done both for biological plausibility, and to evaluate the performance of `xyz` and `glinternet` under less ideal circumstances. Since only `glinternet` assumes the strong hierarchy, this scenario might favour `xyz`.
Based on simulated perturbation matrices $\boldsymbol{X}$, simulated main effects $\beta_i$ and interaction terms $\beta_{i,j}$, we sampled fitness values with $\beta_0 = 0$ according to the fitness landscape model (1.3)

$$\boldsymbol{Y} \sim \mathrm{Norm}\left(\textstyle\sum_i \boldsymbol{X}_i\beta_i + \sum_{i<j}(\boldsymbol{X}_i \circ \boldsymbol{X}_j)\beta_{i,j},\, \sigma^2\boldsymbol{I}\right),$$

where we chose $\sigma^2$ for fixed SNRs $s \in \{2, 5, 10\}$.

### 1.2.5 Evaluation criteria

We focus the evaluation on the estimated parameters of the model, specifically the conditional epistasis terms, $\hat{\boldsymbol{\beta}}_{\{i,j:i<j\}}$, rather than the model's performance in predicting the fitness phenotypes $\boldsymbol{Y}$. Given the ground truth of true conditional epistasis between gene $i$ and $j$, $\boldsymbol{\beta}_{\{i,j:i<j\}}$, we assess the performance of the model to identify epistasis, i.e., estimated non-zero coefficients $\hat{\beta}_{i,j}$, by computing the number of true positives (TPs), false positives (FPs) and false negatives (FN). Here, TPs represent the number of gene pairs $(i, j)$ such that $\beta_{i,j} \neq 0$ and $\hat{\beta}_{i,j} \neq 0$, FPs the number of gene pairs $(i, j) : \beta_{i,j} = 0$ and $\hat{\beta}_{i,j} \neq 0$ and FNs the number of gene pairs $(i, j) : \beta_{i,j} \neq 0$ and $\hat{\beta}_{i,j} = 0$. The performance is then summarised using the following measures

$$\begin{aligned}
\text{precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\
\text{recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\
\text{F1} &= 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}
\end{aligned}$$

Furthermore, we investigate whether estimates $\hat{\beta}_{i,j}$ have the same sign as the ground truth conditional epistasis and we quantify the deviation of the magnitude from the truth. Where applicable, we also evaluate the effect of selection of only those $\beta_{i,j}$ which significantly deviate from zero on the model's performance.

## 1.3 Results

First, we evaluate the proposed approach to estimating epistatic effects from off-target perturbations on simulated data. The approach depends on a model able to detect non-zero pairwise interactions (Figure 1.1). Here, we evaluate the approach using two such alternative models, `glinternet` and `xyz`.
We evaluate the ability of both `xyz` and `glinternet` to identify epistasis between pairs of genes from RNAi screens on simulated data with $p = 100$ genes and $n = 1\,000$ siRNAs. Only for `xyz`, we also test larger data sets, with $p = 1\,000$ and $n = 10\,000$. We use off-target information from real siRNAs and investigate the performance for varying signal-to-noise ratios, number of true interactions, number of observations per double knockdown, and effect sizes for epistasis.
We perform a separate set of tests where we specifically assess the performance of the two methods to identify synthetic lethal interactions, the strongest negative interactions. For this purpose, we simulate a separate data set that contains additional synthetic lethal pairs of genes. In this test, we attempt to identify only lethal interactions using `xyz` and `glinternet`, given increasingly large numbers of genes.

### 1.3.1 Identification of epistasis under varying conditions

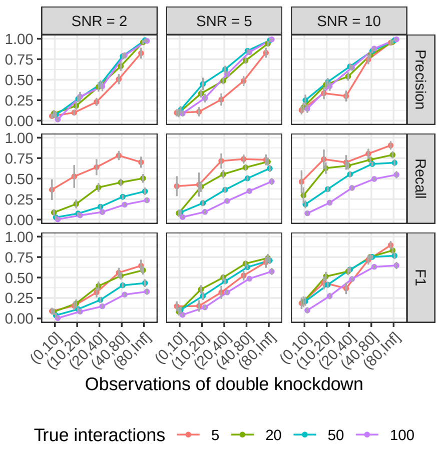
Both `xyz` and `glinternet` are tested on a series of small simulated data sets. For each combination of parameters $q \in \{5, 20, 50, 100\}$, $r \in \{0, 20, 50, 100\}$ and $s \in \{2, 5, 10\}$, controlling the number of true interactions, the number of additional main effects, and the SNRs of the fitness phenotypes, respectively, we sample 50 independent data sets. `xyz` is tested on a series of larger data sets, with parameters $q \in \{50, 200, 500, 1000\}$, $r \in \{0, 200, 500, 1000\}$ and $s \in \{2, 5, 10\}$. Only 10 independent data sets are sampled in these cases. Each data set consists of the perturbation matrix $\boldsymbol{X}$, phenotypes $\boldsymbol{Y}$, true conditional epistasis $\beta_{i,j}$ and main effects $\beta_i$.
The distribution of the number of observations for pairwise knockdowns of gene $i$ and $j$ is shown in Appendix, Figure A.1 for an exemplary perturbation matrix $\boldsymbol{X}$. While only a few genes have many observations, 87% of gene pairs are simultaneously perturbed by at least one siRNA. We also find that number of additional main effects has relatively little impact on detecting interactions (Section A.1), and this value is kept constant during our tests. We select only estimates $\hat{\beta}_{i,j}$ with a magnitude significantly different from zero (q-value $< 0.05$). This significantly improves precision, at a slight cost to recall, using both `glinternet` and `xyz` (Appendix, Figure A.4).

#### Number of double knockdowns per gene pair

We fixed the number of additional main effects to 20 and investigated performance with respect to the number of double knockdowns per epistatic gene pair, i.e. siRNAs that target both genes (Figure 1.2). The results are largely similar for both `xyz` and `glinternet`. As expected, for increasing numbers of observations, we observe an increase in precision and recall with a steeper increase of precision compared to recall and decreased performance for higher number of true interactions. The number of true epistatic gene pairs primarily affects recall, which decreases for higher numbers

of true non-zero $\beta_{i,j}$. For gene pairs with more than 80 observations of the double knockdown, `glinternet` shows strong performance with F1 values between $0.68 - 0.9$ across all tested numbers of true interactions and an SNR larger than or equal to 5 (Figure 1.2a).
`xyz` shows significantly improved performance for gene pairs with more than 40 observations, with F1 values almost all above 0.25. Small numbers of true interactions are particularly accurate, with $F1 > 0.5$ when there are also only 5 such effects (Figure 1.2b).

(a)



(b)

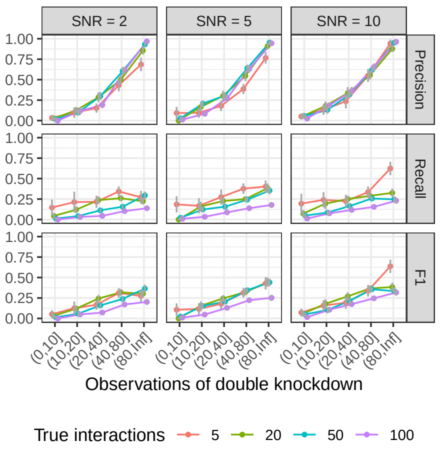


Figure 1.2: Identification of epistasis for increasing numbers of observations of the pairwise double knockdown. The number of additional main effects not overlapping with the set of interacting genes is fixed to 20. Results using (a) `glinternet` and (b) `xyz`.

The number of times each pair of genes is observed is shown in Figure 1.3. We see that in the large simulation, in which all parameters are multiplied by ten, the number of observations of each pair of genes is similarly scaled. As a result, the overall distribution is similar to the smaller simulation.

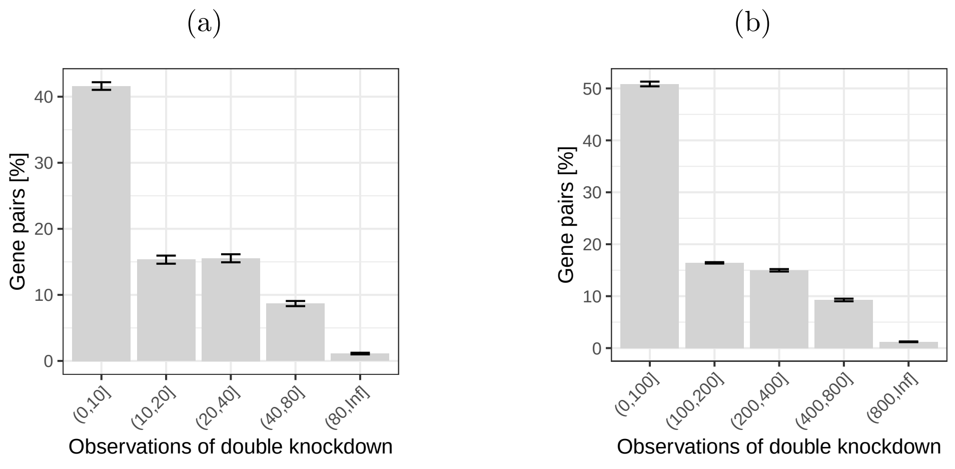


Figure 1.3: The distribution of the fraction of gene pairs stratified by ranges of observed double knockdowns. Gene pairs with zero observations are not shown. (a) $p = 100,\ n = 1000$; (b) $p = 1000,\ n = 10000$.

### Epistatic effect size

We observe that, for both `xyz` and `glinternet`, the performance of the model increases with the absolute value of the magnitude of the conditional epistasis between pairs of genes $|\beta_{i,j}|$ (Figure 1.4). Both for negative and positive epistasis, recall and precision steeply increase with increasing effect size. For pairs of genes with $|\beta_{i,j}| > 1$ and SNRs $\geq 10$, the model performs favourably with F1 values of 0.6 and higher in `glinternet`, and at least 0.25 in `xyz`. Overall performance also marginally improves for `glinternet` at SNR = 5, but no clear effect is seen for `xyz` or SNR = 10. With both `xyz` and `glinternet`, we observe exceptions to the general pattern of the overall V-shape for precision and recall, where strongly negative and positive epistasis and weak epistasis lead to high and low performance of the model, respectively. This effect can be explained by the fact that, after the significance test, an extremely small number of interactions are reported in these ranges (most often only one), with no false positives. The fact that the model's performance notably decreases for small effect sizes around zero explains why we observe a trend of decreasing performance for increasing numbers of true interactions, when we average over all effect sizes. This is because sampling true epistatic effect sizes from Norm$(0, 2)$ for increasing numbers of true interactions increases the fraction of interactions with small effects around zero.
Notably, we can see in Figure 1.4b that even when the overall performance is poor, `xyz` is still able to find a small number of strong interactions relatively accurately. This is particularly promising, since synthetic lethal pairs would be such interactions.

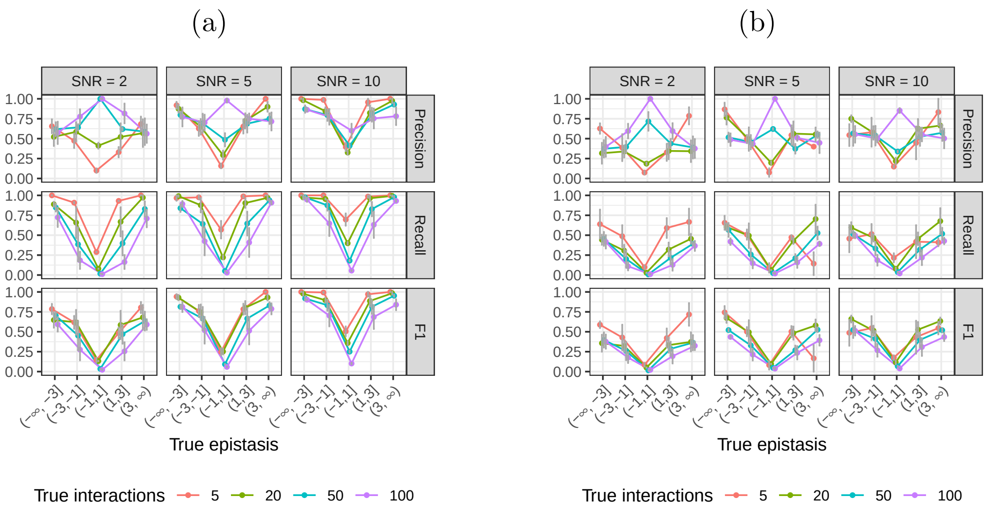


Figure 1.4: Identification of epistasis for varying effect size. Using (a) `glinternet` and (b) `xyz`.

## Direction

We evaluate the ability of each method to distinguish between negative and positive epistasis among epistatic gene pairs identified as true positives (Figure 1.5). For both `glinternet` and `xyz`, the fraction of incorrect estimates of direction (positive vs. negative) is higher for decreasing effect size and increasing number of true interactions. For epistatic effects with an absolute value $> 1$, we observe at most 3% incorrect predictions with `glinternet`, and 8% with `xyz`. We observe at most 9% and 15% incorrect predictions for smaller effect sizes for `glinternet` and `xyz` respectively. Furthermore, we observe that increasing SNRs leads to a subtle decrease of incorrectly predicted direction.

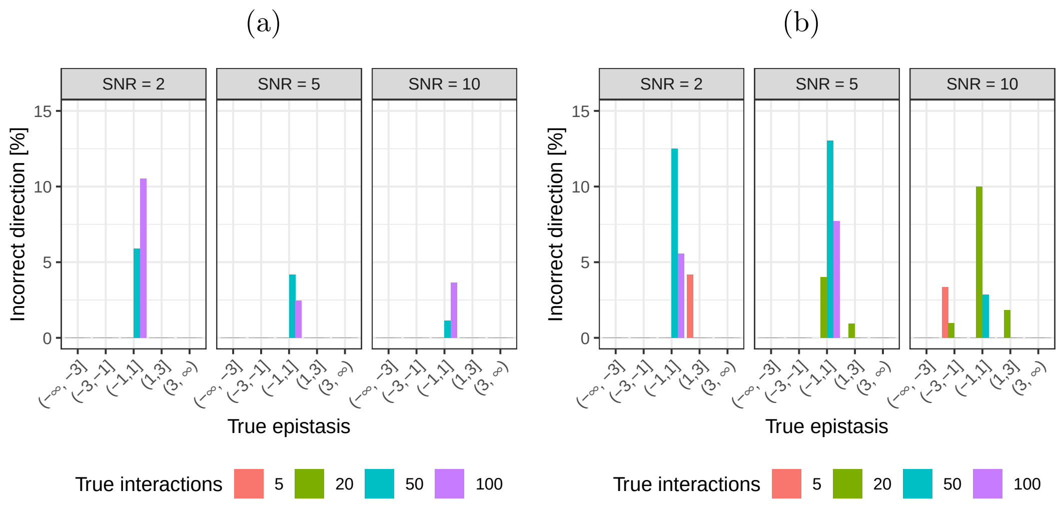


Figure 1.5: Concordance between the sign of true and estimated epistasis. The fraction of incorrectly identified signs between true and estimated epistasis for (a) `glinternet` and (b) `xyz`.

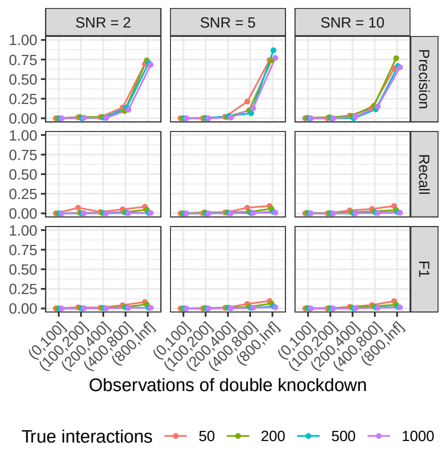
### Magnitude

We evaluate the deviation of the magnitude of estimates for epistasis from the ground truth as a function of observed double knockdowns (Appendix, Figures A.5 and A.12). The deviation in magnitude is computed as $\frac{|\beta_{i,j}|-|\hat{\beta}_{i,j}|}{|\beta_{i,j}|}$, i.e. the percent relative change in deviation with respect to the true epistasis. We observe that across varying numbers of observations the model predicts the magnitude of epistasis between pairs of genes with high accuracy using both `xyz` and `glinternet`.

## 1.3.2 Scalability

Running `glinternet` until it has converged takes a prohibitively long time on larger data sets. While we are able to run our $p = 100, n = 1,000$ simulations in slightly under two minutes, increasing to $p = 1,000, n = 10,000$ takes over two days using ten cores. Since fitting with small lambda values takes the majority of the time, we can improve this by changing the minimum value of lambda that gets used. Adjusting this from $\frac{\texttt{lambdaMax}}{100}$ to $\frac{\texttt{lambdaMax}}{4}$, and fitting only five lambdas in this range rather than fifty, `glinternet` still takes over an hour. This makes the repeated simulations from Subsection 1.3.1 impractical at a larger scale with `glinternet`, although we do investigate some larger data sets in Subsection 1.3.3.
Since `xyz` has significantly shorter run time than `glinternet`, here we more thoroughly investigate performance on larger data sets. Repeating the earlier simulations with every parameter increased by a factor of 10 (Figure 1.6), we find that the overall trends remain the same. The fraction of incorrectly identified signs is omitted, as in this test there are no such results.
There is a significant drop in both precision and recall, and now only effects with a magnitude greater than 3 are found a significant amount of the time (Figure 1.6b).

(a)



(b)

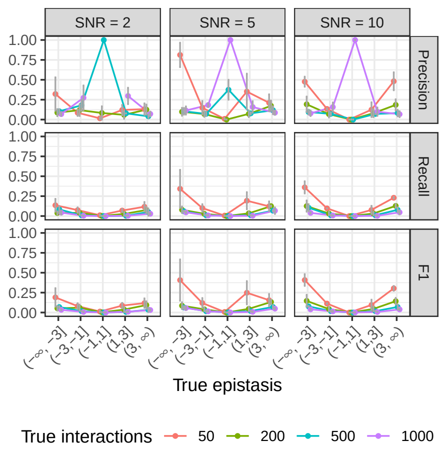


Figure 1.6: Simulations repeated using `xyz` and larger data sets. (a) number of observations of double knockdown. (b) Precision/recall/f1 by actual effect strength.

### 1.3.3 Synthetic Lethal Pairs

Synthetic lethal pairs are of particular interest, and given that `xyz` is able to somewhat reliably find extremely strong interactions, it is natural to ask whether it can be used to quickly find lethal pairs, despite its poor performance on weaker interactions. We fix the number of main effects to 10, and simulated 10 000 siRNAs on 1 000 genes. Synthetic lethal pairs are created as interaction effects of magnitude $-1000$ (log scale). Since lethal pairs often do not have strong main effects (i.e. do not follow the strong hierarchy assumption), the components of the interaction are *not* used as main effects in this case.
Increasing the number of lethal interactions significantly reduces recall, but does not have a clear effect on precision. At this scale, `xyz` is often able to correctly identify some lethal interactions (Figure 1.7), particularly when there are only a few to find.

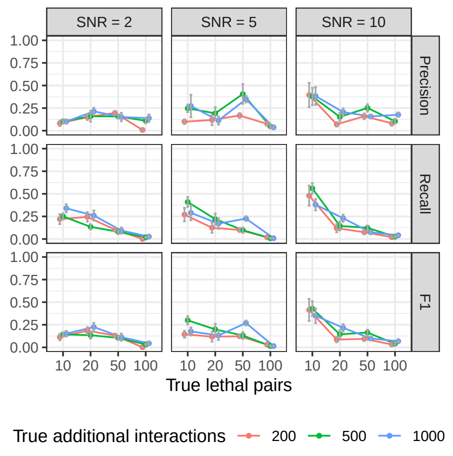


Figure 1.7: Precision, recall, and F1 performance for varying numbers of synthetic lethal pairs, with additional background interactions, using `xyz`. Neither side of the lethal interactions are used as main effects, and as far as lethal interactions are concerned, there is no hierarchy present.

#### Synthetic lethality detection in larger matrices

While we could not run a significant number of tests at this scale using `glinternet`, we could investigate how well its accuracy scales compared to `xyz`. To do this, we simulated sets of up to $p = 4000$ genes, and measured the performance of both `xyz` and `glinternet`. In this case, both to avoid allocating more elements to a matrix than R allows, and to keep the run time of `glinternet` low, only $n = 2 \times p$ siRNAs are simulated. The ratio of siRNAs, genes, main effects, interactions, and lethals, is fixed to: $n = 200$ siRNAs, $p = 100$ genes, $b_i = 1$ main effect, $b_{ij} = 20$ interaction effects, $l = 5$ lethal interactions. Data sets are then generated with these values multiplied by 5, 10, 20, and 40. As in the previous simulation, components of lethal interactions are *not* added as main effects. The strong hierarchy assumption is not valid in this case.

Interactions are then found with both `xyz` and `glinternet`. Here we focus specifically on synthetic lethal detection, and only correct lethal pairs are considered true positives, Any other pair (including a true interaction that is not a lethal) is considered a false positive.

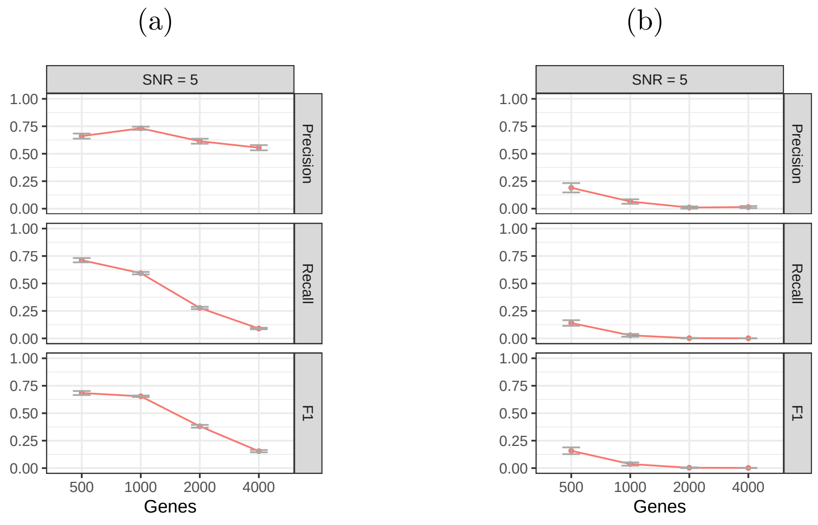


Figure 1.8: The performance of (a) `glinternet` and (b) `xyz` on increasingly large data sets.

We can see in Figure 1.8a that precision with `glinternet` remains fairly consistent as $p$ increases. There is a roughly proportional reduction in recall as the number of lethal interactions increases. After a slight increase from 500 to 100 genes, the actual number of significant interactions found remains fairly consistent. Beyond $p = 2000$, we found that `xyz` typically fails to find any of the lethal pairs (Figure 1.8b)

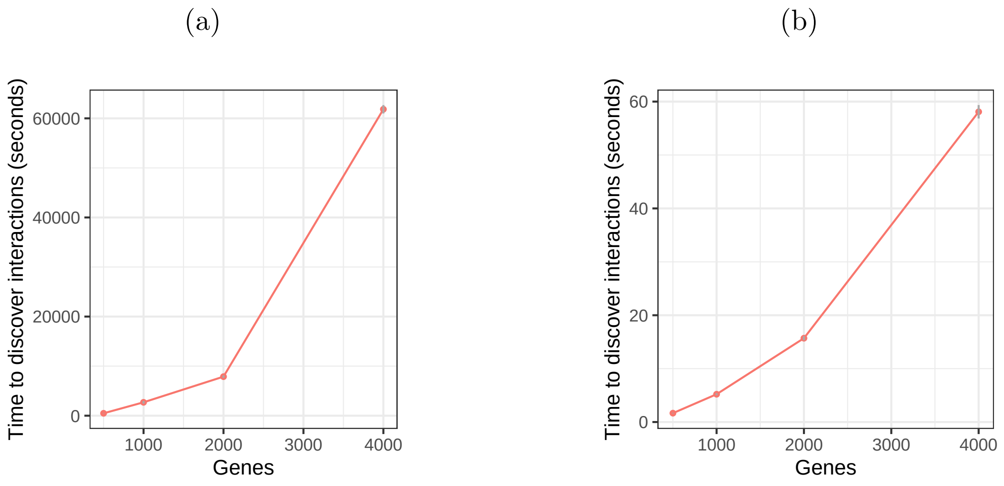


Figure 1.9: Run time in seconds to find interactions on increasingly large data set. 1.9a: `glinternet`. 1.9b: `xyz`. We compiled `glinternet` with OpenMP and ran with `numCores = 10`.

Figure 1.9 shows that neither `xyz` nor `glinternet` quite demonstrate a linear run time, but the run time for `glinternet` increases sharply beyond $p = 2000$. It is possible that this is simply the result of less efficient cache use with larger data, but it is nonetheless worth noting.

### 1.3.4 Violations to model assumptions

For the regularised regression model (1.9) we assume strong hierarchy (1.11) between main effects $\beta_i$ and interaction terms $\beta_{i,j}$ in order to reduce the search space of all possible non-zero coefficients $p + \binom{p}{2}$ during inference. We refer the reader to [53], where Lim and Hastie show how violations to this assumption affect the performance. For instance, the performance of the model is evaluated when the ground truth only obeys weak hierarchy, i.e. only one main effect present, no hierarchy, or anti-hierarchy. Additionally, approximately 2.5% of simulations failed to run using `xyz`, because the estimated interaction frequency of non interacting pairs was too low. These were fairly evenly distributed across all combinations of parameters (Figure A.7), and are not believed to have substantially affected the results.

### 1.3.5 Summary recommendation

After simulating siRNA knockdown data sets of various sizes, and under various conditions, and attempting to reconstruct the interacting pairs using both `xyz` and `glinternet`, we arrive at the following recommendations. For data sets containing less than 4,000 genes (assuming between 2 and 10 experiments per gene), we recommend using `glinternet` to find interactions. Where `glinternet` would have a prohibitively long run time (data sets larger than those mentioned above), `xyz` continues to run quickly, and may still identify some useful results (Figure 1.7). Particularly when one expects a small number of significant interactions, increasing the number of projections beyond $\sqrt{n}$ may improve performance here (see Appendix, Figure A.6c).

### 1.3.6 Effects in real data

Following the recommendation we have arrived at in Section 1.3.5, we apply `glinternet` (followed by a linear regression analysis) to estimate epistatic effects from a real data set. We use the perturbation data from [74], containing siRNA screens targeting kinases of five bacterial pathogens and two viruses, and apply the routine as described in Section 1.2.2 to identify pairwise kinase-kinase interactions. Specifically, we restrict the data to siRNAs that target kinases from the Qiagen Human Kinase siRNA Set V4.1, and the off-target effects within this set, resulting in an input matrix containing 11 214 perturbations $\times$ 667 genes. Using $\mathrm{f} = log_2(\frac{\text{Cells after}}{\text{Cells before}})$ as a fitness measure, we found 1662 effects, 116 of which had a p-value less than 0.05. Since we have assumed that perturbations are binary in our simulations, we continue to do so here. As a result, all non-zero predicted off-target effects are given a value of 1. The ten most significant

predicted effects are shown in Table 1.1. [1] Interestingly, the most significantly associated pair of genes, CDK5R1 and RPS6KA2, are both related to a common pathway. Specifically, CDK5R1 activates CDK5, which, along with RPS6KA2, is part of the IL-6 signalling pathway [44]. Searching both the ConsensusPathDB database [43], and STRING database [86] for relations between the found pairs, we find that a number of the interactions suggested here could be the result of existing known interactions. We each of the identified pairs of genes, we searched for common neighbours (a third gene with which both interact), shared pathways, and whether the produced proteins are found in the same protein complexes, and found the following known relationships:
CDK5R1 and RPS6KA2 share a common neighbour, and are present in four of the same enriched pathway-based sets. TTK and RPS6KA2 share nine common neighbours. RIPK4 and GRK3 share one neighbour,and homologs were found interacting in other species. TNIK and PANK4 share one neighbour, as do MAPK4 and TRPM7, MAP2K6 and UCK1, and HIPK1 and NUAK2. As we could not locate the other identified pairs in the database, we hypothesise that they might constitute novel interactions.

| Gene *i* | Gene *j* | Type | Estimated Effect | P-value |
|---|---|---|---|---|
| CDK5R1 | RPS6KA2 | interaction | 12.52 | 0.0047 |
| RIPK4 | GRK3 | interaction | -3.24 | 0.0056 |
| PHKB | GUK1 | interaction | -7.47 | 0.0061 |
| MAP2K6 | UCK1 | interaction | -40.89 | 0.0094 |
| TNIK | PANK4 | interaction | -37.41 | 0.0115 |
| RPS6KB2 | TTK | interaction | 172.04 | 0.0118 |
| MAPK4 | TRPM7 | interaction | 9.49 | 0.0120 |
| HIPK1 | NUAK2 | interaction | -13.17 | 0.0126 |
| CDK19 | NA | main | 3.80 | 0.0136 |
| C17orf75 | MAPK8IP3 | interaction | 21.74 | 0.0136 |

Table 1.1: Ten most significant predicted effects of siRNA perturbation screens, targeting all human kinases.

For comparison we also fit a linear model including all genes, but no interactions. Comparing the $R^2$ values for each, we find that individual gene effects explain $\approx 15\%$ of the variance ($R^2 = 0.150$) Including the interactions chosen by `glinternet`, and removing the main effects it sets to zero, we have $R^2 = 0.392$, more than doubling the fraction of explained variance. This highlights the importance of accounting for interactions in large-scale genotype-phenotype analyses, and relevance of bioinformatic tools with this capability.

[1]The full set of results, significant or otherwise, can be found at `https://github.com/bioDS/xyz-simulation/blob/master/real_data_results_sorted.csv`

## 1.4 Discussion

To the best of our knowledge, the presented model is the first approach that leverages the combinatorial nature of RNAi knockdown data resulting from sequence-dependent off-target effects for the large-scale prediction of epistasis between pairs of genes. To do this, we take the second-order approximation of the fitness landscape, including only individual and pairwise effects, and attempt to infer the parameters of this model. Since `glinternet` is able to find pairwise interactions among $p = 1,000$ genes, we speculate that searching for three-way interactions is feasible among $\sqrt[3]{1,000^2} = 100$ genes. We are not aware of any software currently able to do this, however.
For the majority of our tests, we simulate the presence of a strong hierarchy. This constraint would imply that for the inference of non-zero epistatic effects between gene $i$ and $j$, $\beta_{i,j}$, we penalise cases where the main effects for both single genes, $\beta_i$ and $\beta_j$, are zero. This constraint significantly decreases the complexity of the search space of interactions. However, in biology there are many examples of epistasis where the marginal effects of individual genes are very small, for instance if both genes redundantly execute the same function within the cell [71]. Costanzo et al. [17] found in their study of experimental double knockouts in yeast that single mutants with decreasing fitness phenotypes tended to exhibit an increasing number of genetic interactions. This observation is reassuring for `glinternet`, which can pick up the interaction as long as the true single-mutant effects are not exactly zero. Moreover, Lim and Hastie showed in a simulation study that the model is in fact flexible enough to also identify pairwise interactions violating the strong hierarchy constraint [53]. For the detection of strong interactions, specifically synthetic lethal pairs, we have also demonstrated that the strong hierarchy constraint is not required.
In a simulation study, we sampled perturbations for $n = 1000$ siRNAs and $p = 100$ genes, and $n = 10000$ siRNAs with $p = 1000$ genes. As a consequence of high-throughput genome-wide screening platforms, the setting of $n = 10 \times p$, i.e. ten perturbations with different siRNAs per gene, is realistic even for higher order organisms with tens of thousands of genes [74, 78]. Sampling the perturbations directly from commercially available RNAi libraries allows us to translate results from the simulation study to applications on real data. We observe that increasing SNRs, as expected, results in an overall increase of the number of successfully identified gene pairs with true epistasis.
Nevertheless, we found that even for a moderate SNR of only 2, the model identifies interactions with acceptable performance using `glinternet` (F1 $> 0.5$ for 50 true interactions), when we observed each double knockdown over 40 times (Figure 1.2a) or the effect size of epistasis is larger than 1, i.e. $|\beta_{i,j}| > 1$ (Figure 1.4a). For an SNR of 5 and across all tested numbers of additional gene pairs and epistatic effect sizes, the performance of the model is approximately constant at around F1 $= 0.5$, independent of the number of true epistatic gene pairs (Figure A.2b).

Performance in our simulations also suggests that `xyz` is unable to accurately identify interactions in large data sets. Although `xyz` has a consistently short run time, and appears capable of running on genome-scale data, we see a significant drop in all other performance measures beyond $p = 1000$ genes.

The results when using `glinternet`, however, suggest that the general approach is able to accurately identify pairwise epistasis from large-scale RNAi data sets, given that the SNR of measured fitness phenotypes is larger than 2 and the effect size of epistasis is larger than 1. It is challenging to compare the performance of these models to approaches that estimate genetic interactions from other data, such as for instance from double knockout experiments [17], due to different scales of the epistatic effect size, however, the high precision of `glinternet` seems quite competitive. Moreover, our simulations demonstrated that if true epistatic effects between pairs of genes are identified, the model identifies both the direction of epistasis (positive and negative) as well as the magnitude of the epistatic effect with high accuracy (Figures 1.5 and A.5). In detecting lethal interactions specifically, the high precision of `glinternet` after testing for significant deviations is particularly promising. Using this as a method to detect likely synthetic lethal interactions from RNAi data sets, we could propose candidates for further investigation as anticancer drug targets [10][3]. While the run time may prevent `glinternet` from being used as such a method in genome-scale applications, we can recommend it for use with smaller data sets, or where the number of potential interactions can be significantly reduced prior to running `glinternet`. As the precision does not appear to suffer with larger input, only the run time, we believe combining linear regression with a perturbation matrix is a promising method for further investigation, and work to improve the performance sufficiently for use in genome-scale applications is ongoing.

Finally, it is worth noting that this approach is not limited to siRNA perturbation matrices, or to synthetic lethal detection. Any method of suppressing gene expression, combined with an affected proxy for fitness, could be used to find likely candidates for epistasis.

# Chapter 2

# A More Scalable Lasso-based Method and Software for Inferring Pairwise Interactions from Perturbation Screens

We saw in Chapter 1 that a lasso-based approach to inferring interactions from an siRNA perturbation matrix is a feasible method for large-scale interaction detection. Attempting to use `glinternet` at such a scale, we run into scalability problems due to the running time, however. Finding interactions in an siRNA screen of $1,000$ genes with ten siRNAs per gene takes several days using ten cores on an Opteron 6276. Our aim is to fit an even larger model, including all $p \approxeq 20,000$ human protein-coding genes, with as many as $n = 200,000$ siRNAs. Doing so requires the development of new methods and software. Our aim in this chapter is to develop a method able to fit such a model in under three days, maintaining performance without requiring more than one Terabyte of memory,
We have developed an R-package that is able to perform lasso regression on all pairwise interactions on the same one thousand gene screen in twenty seconds, and is able to fit a genome-scale data set with $27,000$ genes and $30,000$ siRNAs in several hours. This is made possible by taking into account some details of our input data, specifically, that our input matrix $X$ is both sparse and strictly binary. To perform lasso-based regression on this matrix, we begin with an existing fast algorithm, parallelise it, and adapt it for use on our binary perturbation matrices. In this chapter we provide a detailed explanation of this implementation, followed by some analysis of its scalability.

## 2.1 Lasso-based Method for Inferring Pairwise Interactions

### 2.1.1 Input Data

The input to our package is the same as in Chapter 1. We take a as our input a perturbation matrix $\mathbf{X}$ and fitness vector $\mathbf{Y}$. $\mathbf{X} \in \{0,1\}^{n \times p}$ is a binary matrix, where $x_{ij} = 1$ if gene $j$ is knocked down by siRNA $i$, 0 otherwise. The entry $y_i$ of $\mathbf{Y} \in \mathbb{R}^n$ is the approximation of fitness in the presence of siRNA $i$. For each column $X_j$ of $\mathbf{X}$ we attempt to find a value $\beta_j$ that best explains the effect gene $j$ has on fitness. Ideally, for every row $i$ of $\mathbf{X}$, the sum of these values $\sum_{j=1}^{p} x_{ij}\beta_j$ would be exactly the measured fitness effect $y_i$ (see Figure 2.1 for an example). Given $\mathbf{X}$ and $\mathbf{Y}$, we aim to find effect sizes $\beta$ that minimise the errors $\mathcal{E}$ shown in the formulas of Figures 2.1, 2.2. Note that in at this point we do not consider interactions.

Effect of Gene $1 \dots p$ being knocked down

Gene 1 Gene 2 Gene p

$$\beta_1 x_{1,1} + \beta_2 x_{1,2} + \ldots + \beta_p x_{1,p} + \mathcal{E} = y_1$$

Fitness measure after siRNA knockdowı

Figure 2.1: Single siRNA effects

Effect of Gene $1 \dots p$ being knocked down

Gene 1 Gene 2 Gene p

$$\beta_1 x_{1,1} + \beta_2 x_{1,2} + \ldots + \beta_p x_{1,p} + \mathscr{E}_1 = y_1$$

$$\beta_1 x_{2,1} + \beta_2 x_{2,2} + \ldots + \beta_p x_{2,p} + \mathscr{E}_2 = y_2$$

$$\vdots$$

$$\beta_1 x_{n,1} + \beta_2 x_{n,2} + \ldots + \beta_p x_{n,p} + \mathscr{E}_n = y_n$$

Fitness measure for siRNA $1 \dots n$

Figure 2.2: Matrix of siRNA effects

As in the previous chapter, our goal is to estimate both the effects $\beta_1, \ldots, \beta_p$ of knocking down each gene, and the effects $\beta_{1,2}, \ldots, \beta_{p-1,p}$ of knocking down pairs of genes simultaneously. To do this, we add a column for each pair of genes, converting the siRNA matrix $\mathbf{X} \in \{0,1\}^{n \times p}$ into the pairwise matrix $\mathbf{X}_2 \in \{0,1\}^{n \times p'}$, where $p' = \frac{p(p+1)}{2}$. This model includes all pairwise interactions, and fitting it is equivalent to finding pairwise epistasis in Chapter 1.
We construct the matrix $\mathbf{X}_2$ as follows. For every column $i$ from 0 to $n$, we take every further column $j$ from $i+1$ to $n$, and form a new column by taking the bit-wise *and* over all elements of the columns $i$ and $j$ (Figure 2.3).

$$\underset{x_a}{\begin{bmatrix}1\\0\\1\\0\\1\end{bmatrix}} \wedge \underset{x_b}{\begin{bmatrix}1\\1\\0\\0\\1\end{bmatrix}} \rightarrow \underset{x_{a,b}}{\begin{bmatrix}1\\0\\0\\0\\1\end{bmatrix}}$$

Figure 2.3: Creation of pairwise siRNA effect columns

This gives us the complete pairwise matrix $\mathbf{X}_2$, as shown in Figure 2.4.

$$\overset{\text{Gene 1}}{\beta_1 x_{1,1}} + \ldots + \overset{\text{Gene } p}{\beta_p x_{1,p}} + \overset{\text{Genes 1 and 2}}{\beta_{1,2} x_{1,1\wedge 2}} + \ldots + \overset{\text{Genes 1 and } p}{\beta_{1,p} x_{1,1\wedge p}} + \ldots + \overset{\text{Genes } p-1 \text{ and } p}{\beta_{p-1,p} x_{1,p-1\wedge p}} + \mathscr{E}_1 = y_1$$

$$\beta_1 x_{2,1} + \ldots + \beta_p x_{1,p} + \beta_{1,2} x_{2,1\wedge 2} + \ldots + \beta_{1,p} x_{2,1\wedge p} + \ldots + \beta_{p-1,p} x_{2,p-1\wedge p} + \mathscr{E}_2 = y_2$$

$$\vdots$$

$$\beta_1 x_{n,1} + \ldots + \beta_p x_{n,p} + \beta_{1,2} x_{n,1\wedge 2} + \ldots + \beta_{1,p} x_{n,1\wedge p} + \ldots + \beta_{p-1,p} x_{n,p-1\wedge p} + \mathscr{E}_n = y_n$$

Figure 2.4: Matrix of Pairwise siRNA effects

### 2.1.2 Cyclic Linear Regression

Our approach to lasso regression is based on a cyclic coordinate descent algorithm from Fu [26], as described in Wu and Lange [100]. This method begins with $\beta_j = 0$ for all $j$, and updates the beta values sequentially, with each update attempting to minimise the current total error. Here this total error is the difference between the effects we have estimated and the fitness we observe, given the genes that have been knocked down. Where $y_i$ is the $i$th element of $\mathbf{Y}$, $\beta_j$ is the $j$th element of $\beta$, and $x_{ij}$ is the entry in the matrix $\mathbf{X}_2$ at column $j$ of row $i$, the error is the following.

$$\sum_{i=1}^{n} |y_i - \sum_{j=1}^{p'} x_{ij} \cdot \beta_j| \tag{2.1}$$

Note that we assume that the fitness vector $\mathbf{Y}$ is already centred around 0, and omit the offset $u$ present in Wu and Lange [100]. In the context of pairwise genetic interactions we would rather have a smaller number of definitely relevant effects, than a large number of marginally relevant ones. To this end, we add the lasso penalty to the error in Equation 2.1. This penalises large beta values according to a parameter $\lambda$, and results in a smaller set of, typically larger, non-zero beta values [90]. With this added penalty we minimise the value:

$$\sum_{i=1}^{n} |y_i - \sum_{j=1}^{p'} x_{ij} \cdot \beta_j| + \lambda \sum_{j=1}^{p'} |\beta_j| \tag{2.2}$$

We do this by minimising the component of this error that each $\beta_j$ is able to account for. For a particular $\beta_k$, the component of this error that we can minimise by changing this value is:

$$f(\beta_k) = \sum_{i=1}^{n} |x_{ik} \cdot (y_i - \sum_{j=1}^{p'} x_{ij} \cdot \beta_j)| + \lambda|\beta_k| \tag{2.3}$$

This error comes from the non-zero entries in column $k$ of $\mathbf{X}$. Since in our case all entries are either 1 or 0, this is simply the sum of errors of rows where column $k$ has an entry, with a penalty imposed for large beta values.
To minimise this component $f(\beta_k)$ alone, we define $r_i$ and $S_k$:

$$r_i = \sum_{j=1}^{p'} x_{ij} \cdot \beta_j$$
$$S_k = \sum_{i=1}^{n} x_{ik}$$

The error affected by a single beta value (Equation 2.3) can then be minimised by increasing $\beta_k$ by the following:

$$\Delta\beta_k = \begin{cases} min(0, \beta_k + \frac{\sum_{i=1}^{n}(x_{ik}(y_i - r_i))}{S_k} - \lambda) & \text{for } \beta_k + \frac{\sum_{i=1}^{n}(x_{ik}(y_i - r_i))}{S_k} > 0 \\ max(0, \beta_k + \frac{\sum_{i=1}^{n}(x_{ik}(y_i - r_i))}{S_k} + \lambda) & \text{for } \beta_k + \frac{\sum_{i=1}^{n}(x_{ik}(y_i - r_i))}{S_k} < 0 \end{cases} \tag{2.4}$$

This is equivalent to the solution from Wu and Lange [100], as we will now show. Their solution is defined separately for positive and negative $\beta_k$:

$$\beta_{k-} = min\{0, \beta_k - \frac{\frac{\delta}{\delta\beta_k} g(\theta) - \lambda}{\sum_{i=1}^{n} x_{ik}^2}\}$$

$$\beta_{k+} = min\{0, \beta_k - \frac{\frac{\delta}{\delta\beta_k} g(\theta) + \lambda}{\sum_{i=1}^{n} x_{ik}^2}\}$$

$$\text{Where } \frac{\delta}{\delta\beta_k} g(\theta) = -\sum_{i=1}^{n} q_i x_{ik},$$
$$\text{and } q_i = y_i - u - \sum_{j=1}^{p'} x_{ij}\beta_j$$

Note that we assume the intercept term $u = 0$, because $Y$ is centred around 0, and $u$ can therefore be omitted. We shall first focus on proving the equivalence of our construction for $\beta_{k-}$. Since $x_{ik} \in \{0, 1\}$, it follows $x_{ik} = x_{ik}^2$, and therefore $\sum_{i=1}^{n} x_{ik}^2 = S_k$. This gives us

$$\beta_{k-} = min\{0, \beta_k - \frac{\frac{\delta}{\delta\beta_k}g(\theta) - \lambda}{S_k}\}$$

Also substituting $\frac{\delta}{\delta\beta_k}g(\theta)$, we have:

$$\beta_{k-} = min\{0, \beta_k - \frac{-\sum_{i=1}^{n} x_{ik}(y_i - r_i) - \lambda}{S_k}\}$$

$$= min\{0, \beta_k + \frac{\sum_{i=1}^{n} x_{ik}(y_i - r_i) + \lambda}{S_k}\}$$

$$= min\{0, \frac{S_k\beta_k + \sum_{i=1}^{n}(y_i - r_i) + \lambda}{S_k}\}$$

This is equivalent to Equation 2.4 for $\beta_k < 0$. The positive solution is equivalent, substituting *min* for *max* and subtracting $\lambda$. Iteratively minimising beta values until the solution converges, we have Algorithm 1. We consider the algorithm to have converged when $\frac{e_{prev}}{e_{after}} < t$ for some threshold $t$, where $e_{prev}$ is the error before the iteration and $e_{after}$ the error after the iteration. We arbitrarily choose $t = 1.0001$ as the default in our software.

**Algorithm 1:** Sequential Cyclic Algorithm

```
while not converged do
    foreach $\beta_k$ do
        $\Delta\beta_k \leftarrow \frac{\sum_{i=1}^{n}(x_{ik}(y_i - r_i))}{S_k}$;
        if $|\beta_k + \Delta\beta_k| > \lambda$ then
            if $\beta_k + \Delta\beta_k > 0$ then
                $\beta_k \leftarrow \beta_k + \Delta\beta_k - \lambda$;
            end
            else
                if $\beta_k + \Delta\beta_k < 0$ then
                    $\beta_k \leftarrow \beta_k + \Delta\beta_k + \lambda$;
                end
            end
        end
    end
end
```

A naive implementation would read every entry of the $\mathbf{X}_2$ matrix, and every value in the vector $\mathbf{Y}$, every iteration, for every beta update. With $\mathbf{X}_2 \in \{0,1\}^{n \times p'}$, $\beta \in \mathbb{R}^{p'}$ and $\mathbf{Y} \in \mathbb{R}^n$, this is $\Theta(np^4)$ operations per iteration. Since $x_{ij}$ and $y_i$ are constant, $r_i = \sum_{j=1}^{p} x_{ij}\beta_j$ only changes when $\beta_j$ changes. Updating this value every iteration, rather than re-calculating it, we only need to perform $n$ operations per $\beta$ update. This brings the number of operations for an iteration down to $np^2$. To update $\beta_j$, we now read a single column, $X_j$, and the values of $r_i$ and $Y_i$ for each non-zero entry $x_{ij}$. By including $\mathbf{Y}$ in $\mathbf{R}$ such that $r_i = y_i - \sum_{j=1}^{p} x_{ij}\beta_j$ for all $i$, we no longer need to read $\mathbf{Y}$. We can further reduce the work that needs to be done by storing a sparse representation of $\mathbf{X}_2$. While $\mathbf{X}$ is a sparse matrix, $\mathbf{X}_2$ is an extremely sparse matrix. In a typical simulated data set from Chapter 1, we go from, on average, 112 out of $1,000$ entries per column in the $\mathbf{X}$ matrix to 16 out of $1,000$ on average in $\mathbf{X}_2$. We therefore store $\mathbf{X}_2$ in the following format. Each column is a list of the positions of its non-zero entries (Figure 2.5). Since these are one by definition, we don't store their value. We store the matrix column-wise to ensure the column $X_j$ can be read quickly when updating $\beta_j$. Each column is therefore stored as a separate array of integers.

$$X : \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 3 & 1 & 0 & 2 \\ & & 2 & \\ & & & \\ & & & \end{bmatrix}$$

Figure 2.5: Simple sparse matrix representation

As a result of the memory and run time requirements, this implementation is usable up to $p = 1,000$ but not for values as large as $p = 20,000$. To overcome this, we compress the matrix, and parallelise the beta updates.

### 2.1.3 Compression

To reduce memory usage and the time taken to read each column with larger input data, we compress the columns of $\mathbf{X}_2$. Because we read the columns sequentially, we replace each entry with the offset from the previous entry. This reduces the average entry to a relatively small number, rather than the mean of the entire row. These small integers can then be efficiently compressed with any of a range of integer compression techniques (Figure 2.6), a subject that has been heavily developed for Information Retrieval. We compare a number of such methods, including the Simple-8b algorithm from Schlegel, Gemulla, and Lehner [77] which we implement and use in our software.

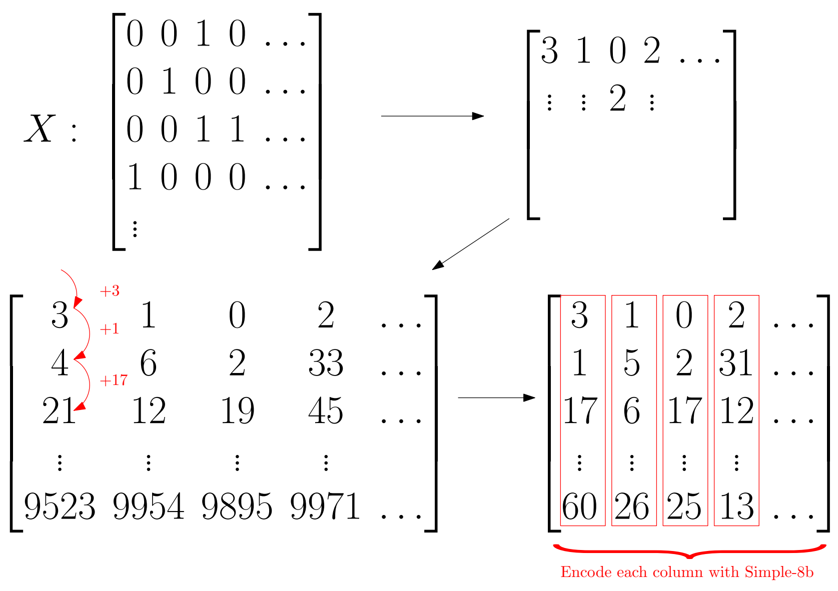


Figure 2.6: Compression of the sparse $\mathbf{X}_2$ matrix.

## Simple-8b

Simple-8b is a non-SIMD compression scheme, with performance comparable to other state of the art methods [77, 57, 92]. While SIMD-based compression schemes can offer significantly improved compression and decompression speed [49] [77], their implementation is architecture dependant. Simple-8b only requires a CPU be able to efficiently handle 64-bit arithmetic. This amounts to requiring a modern CPU, which we assume is already a requirement for processing large data sets.
Simple-8b is a 64-bit variation of the Simple-9 encoding scheme [2], and stores a sequence of integers in a single 64 bit word. The number of integers stored depends on the size of the largest one, and is indicated by a four bit 'selector'. The remaining 60 bits are divided into integers of size $1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 15, 20, 30$ or 60, with between 240 (only possible if all values are zero) and one integer stored. As seen in Figure 2.7, this considerably reduces the size of $\mathbf{X}_2$ in our test data (two sets from Chapter 1, one with $p = 100$, $n = 1,000$, another with $p = 1,000$, $n = 10,000$). In the larger $p = 1,000$ set, total memory use is reduced by over 85%.

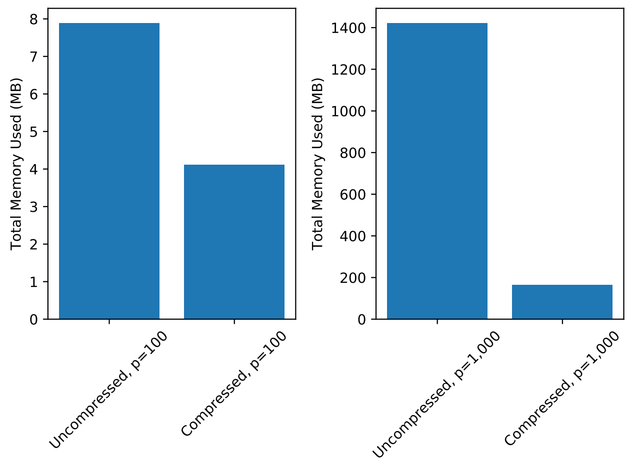
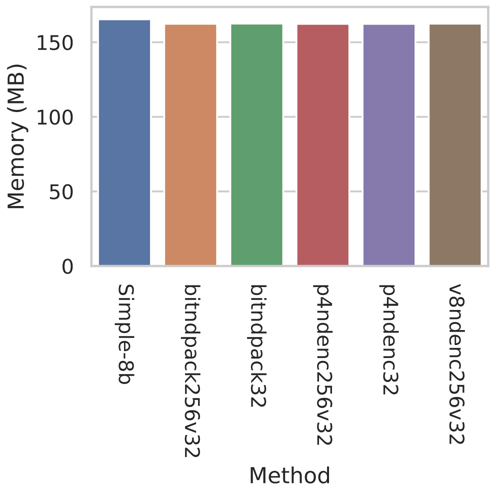


Figure 2.7: Compression effect on memory use. Note that this is the total peak memory use of the program, not solely the memory used by the matrix $\mathbf{X}_2$. In both cases $n = 10 \cdot p$.

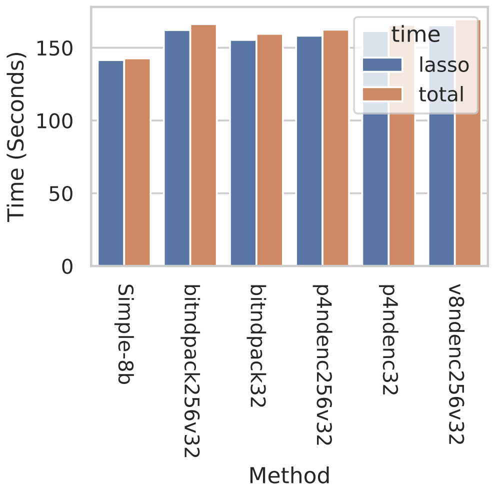
It is worth noting that this compression works well even for non-sparse sections of the matrix, since the offsets are extremely small. In an extreme case, we can store up to 240 sequential 1's in a single 64-bit word. That is to say that, while the earlier offset-based format (Figure 2.5) relies on the sparsity of the matrix for its efficiency, compression works well regardless.

### Comparing Compression Methods

While Simple-8b allows our implementation to be used on any 64-bit CPU, we could also take advantage of SIMD-based methods where the such CPU instructions are available. To determine whether this is a worthwhile improvement, we compare our Simple-8b implementation to a number of state of the art alternatives.
Recent work suggests TurboPFor [70] has a particularly high compression ratio [92]. We therefore compare the best performing methods from TurboPFor against our implementation of Simple-8b (Figure 2.8). The tests are performed using 32 threads across two eight-core (16 SMT threads) Intel(R) Xeon(R) Gold 6244 CPUs in a NUMA system.
To compare these methods, we perform 50 regression iterations on a test data set of $p = 1,000$ genes and $n = 10,000$ siRNAs. We examine the total time taken for the process, as well as the total memory used and time for the regression function alone (excluding calculating and compressing the interaction matrix).



(a)



(b)

Figure 2.8: Comparison of Compression Methods. (a) Total memory used, compressing the sparse $\mathbf{X}_2$ matrix with each method. (b) Total time taken and time taken (including compressing $\mathbf{X}_2$) and time taken for lasso regression alone, using each method.

We see that both the time to produce the compressed matrix (seen in Figure 2.8 as the difference between total time and lasso-only time), and the run time are comparable for all TurboPFor methods.[1] While every TurboPFor method we tested improved the compression ratio compared to Simple-8b (Figure 2.8a), we consistently found that the run time was longer (Figure 2.8b). It is possible that this is a result of the way the columns are being read in each method. Using TurboPFor, we compress and decompress entire columns at a time. With our Simple-8b implementation, we process each 64-bit word separately. This allows us to use the column as it is being decompressed. Avoiding re-reading the column after decompression also allows the entries to be evicted from the cache earlier.
While it is also possible to process compressed words as they are read using the tested TurboPFor methods, there does not appear to be a significant enough difference in compression to justify the restriction to SIMD-capable CPUs, or the complications in the build system that arise using a separate compression library.

## 2.2 Choosing Lambda

The lasso penalty includes a parameter, lambda. This parameter determines the extent to which we penalise large beta values, and can range from allowing all values ($\lambda = 0$) to allowing only zero, ($\lambda \to \infty$). Choosing the correct value of lambda is essential if we want to include only the significant effects. This is typically done by choosing an

[1]The compression time is not comparable for all methods. Our Simple-8b implementation compresses columns in parallel, whereas TurboPFor does not. Regression is done in parallel using all cases, using the method described in Subsection 2.4.4.

initial value sufficiently large that all beta values will be zero, and gradually reducing lambda, fitting the model for each value [25]. At each new lambda value we can carry over the beta values from the previous iteration. This way, only the effects that are now allowed, but were not before, need to be found. The time taken to fit a decreasing sequence of lambda values in this way is not much more than it would take to fit the smallest on its own, and in some cases is even faster [25].

Choosing the stopping point, the lowest value of lambda, is often done by placing a limit on the number of values that will be used (50 or 100 are typical), and choosing the best among these with cross-validation [100, 25]. In our case, we do not rely on lambda alone to decide which effects are significant. Instead we only use lasso regression to choose the non-zero beta values, and then perform Ordinary Least Squares (OLS) regression, including only the columns of $\mathbf{X}_2$ corresponding to these values. The p-values from OLS regression are used to determine whether an effect is significant. It suffices then to continue decreasing lambda until a predefined arbitrary small lambda value, relying on OLS regression to filter for significant results. With many non-zero beta values, a result of sufficiently small lambda, the OLS regression step can take longer than the initial lasso regression, however. The small values of lambda also account for the majority of the lasso run time, and if we can stop the lambda sequence early, this can be significantly improved.

We provide two options for choosing lambda in our package. First, we choose lambda such that the number of non-zero effects is small enough for OLS regression. Second, we implement a fast method for empirically choosing a reasonable stopping point. More specifically, we choose lambda in the following way. We begin with lambda sufficiently large that all beta values will be zero. Lambda is then gradually decreased, setting the new value at each step to $\lambda_{new} = 0.9 \cdot \lambda_{prev}$. We decrease lambda until we reach or pass the minimum value (0.05 by default). After fitting with each lambda, we optionally check two stopping conditions. First, we check if a maximum number of non-zero beta values is set, and whether we have reached this value. Second, if enabled, we perform the adaptive calibration test [13], stopping if the conditions are met.

We use the adaptive calibration lambda selection method from Chichignoud, Lederer, and Wainwright [13] instead of the standard $K$-fold cross-validation because cross-validation requires fitting each lambda value $K$ times, and this increase to the run time is unacceptable for large data. Adaptive calibration only requires a single relatively small calculation for each lambda. It aims to choose the minimum value of lambda that is sufficient to control fluctuations. Assuming $\mathbf{X}_2$ satisfies the design condition from Van de Geer [93], the value chosen is within a constant factor of this ideal value, and precision and recall are comparable to cross-validation [13].

We compare precision, recall, and running time when using the adaptive calibration stopping condition to running as many iterations as we can, on small and large simulated data sets from Chapter 1. In all cases, we then perform the OLS regression step and filter out results with a q-value $\geq 0.05$. Using the smaller set ($p = 100,\ n = 1,000$), we run until reaching our lower limit of $\lambda = 0.05$. With the larger data set this may include too many columns for OLS regression, so we limit the number of non-zero beta values to $2,000$. The results of this test can be seen in Figure 2.9 and Figure 2.10.

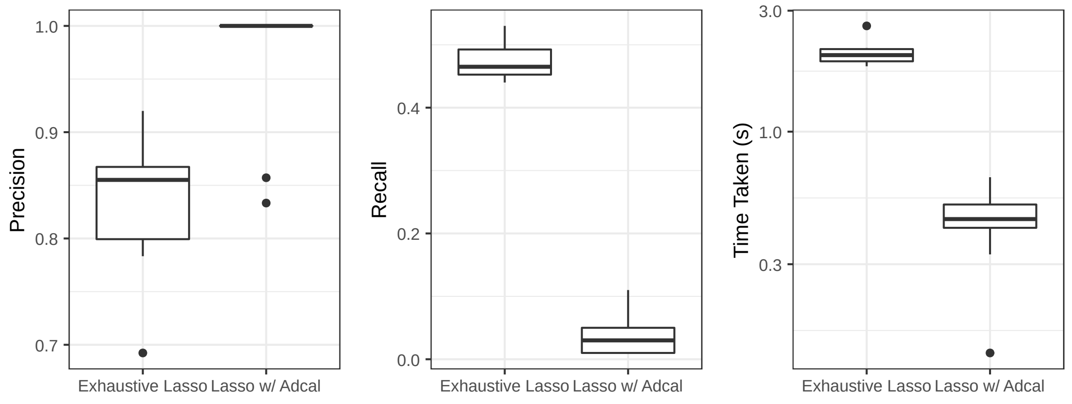


Figure 2.9: Adaptive calibration effect on small sets. 'Exhaustive Lasso' runs until the predefined minumim lambda value of 0.05 is reached. 'Lasso w/ Adcal' halts once adaptive calibration conditions are met, continuing until the cutoff otherwise.

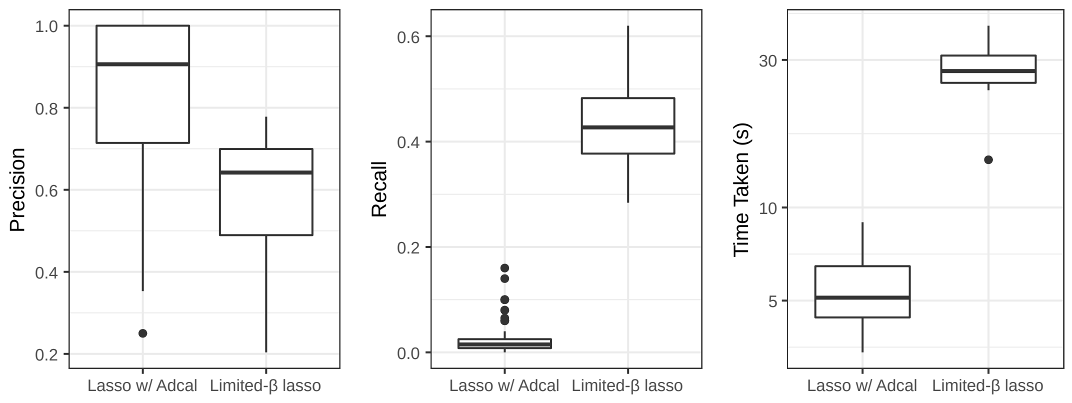


Figure 2.10: Adaptive calibration effect on large sets. 'Limited-$\beta$ Lasso' runs until $2,000$ beta values are non-zero, continuing until the cutoff otherwise. 'Lasso w/ Adcal' halts once adaptive calibration conditions are met, continuing until the cutoff otherwise.

We see significantly higher recall when running until a very small cutoff, and filtering the results with ordinary least squares afterwards. The number of variables included in the latter regression limits the usefulness of this approach, however. We find that for large data sets there are often too many non-zero betas, all of which must be included when performing OLS regression. We therefore include the option to halt regression after a certain number of non-zero betas have been set.
Both the adaptive calibration option and limiting non-zero betas can prevent the algorithm from finding small effects. This is in many cases beneficial, as it is the small values of lambda, and therefore the small effects, that are computationally expensive. Moreover, our previous benchmarks (Chapter 1) show that small effects are also the least likely to be correctly identified. With that in mind, we do not consider ignoring these effects to be a problem.

## 2.3 Fault Tolerance

There are between 20 and 30 thousand protein coding genes [14], and running our method on matrices with this many columns can take over 20 hours. If the job is interrupted, fails for some reason, or needs to be repeated, we can avoid repeating the entire process by keeping a log of the running job. We do this in the following way: Either every iteration, every lambda change, or never (depending on the user's choice), we save the current lambda values to a log. The log format is shown in Figure 2.11, with an example in Figure 2.12

```
[status]
[run args]
[human-readable format explanation]
[n, p, num_beta]
[w/ ][current iteration] [current lambda]
[current beta values]
[w/ ][current iteration] [current lambda]
[current beta values]
```

Figure 2.11: Log file format

```
finished
./build/utils/src/lasso_exe testX.csv testY.csv int F 100
 ↪  1000 100 -1 1 t l
lasso log file. metadata follows on the next few lines.
 The remaining log is {[ ]done/[w]ip} $current_iter,
  ↪  $current_lambda\n $beta_1, $beta_2 ... $beta_knum_rows,
  ↪  num_cols, num_betas
1000, 100, 5050
␣00051, 00050, +5.726417e-01
```

Figure 2.12: Example log file

The entries of the log are as follows.

- *status:* Current status of this run of the program. Either `still running` or `finished`. We only resume from a log if it is still running (i.e. the run did not complete) and the arguments are identical in the log and the current run.

- *run args:* The arguments the program was given for this run. This is used to decide whether the log represents a previous attempt to run on the same matrices. If these arguments are identical to the those of the currently running program, and the log indicates it did not complete, then we attempt to resume from the last entry in the log.

- *human-readable format explanation:* A brief explanation of the format for the sake of a user inspecting the log.

- *n, p, num-beta:* In order, the number of rows in the matrix, number of columns, and number of beta values we are trying to solve for. *num-beta* will either be $p$ or $\frac{p \cdot (p+1)}{2}$, depending on whether we are solving for main effects or interactions respectively.

- *w/:* The beginning of an entry into the log, either after an iteration, or finishing a lambda, depending on settings. Either `` `w' `` or `` `  ' ``, depending on whether this entry is currently being written, or is a complete entry. If the value is `` `w' ``, we should not attempt to restore from this entry, as it may be incomplete. In particular, since the entry is written over a previous one at the same point in the file, later sections may be leftover from such an entry, rather than being part of this one.

- *current iteration:* The number of the iteration (of the current lambda) saved in this entry. This is always a five digit integer, with leading zeros as necessary.

- *current lambda:* The lambda value being processed as of this entry. This is always a seven significant figure number in scientific notation, with a leading `` `+' `` or `` `-' `` sign. This guarantees every entry will be exactly 13 characters.

- *current beta values:* A comma separated list of the current beta values, also as seven significant figure scientific notation numbers. Since each entry is 13 characters, a comma, and a space, and ends with a newline character, we can guarantee the length of the line to be $15 \cdot |\beta| + 1$.

In case an entry is interrupted while being written, we should also keep a copy of the previous entry. The log will always include exactly two entries (once it has run long enough to save at least twice), and we alternate which one is written each time we save to the log. This avoids creating an excessively large log, with entries from every previous iteration or lambda value, while still guaranteeing we can restore from the previous entry in case the process is interrupted while writing. Since there is no way to distinguish between the old and new characters in the entry if writing is interrupted, we need the beginning `'w'`.
Note that regardless of the current iteration, lambda, or beta values, every entry will be the exact same number of characters. This is to guarantee that the second entry

always begins after the same number of bytes. We can then overwrite only the second entry.

## 2.4 Parallelisation

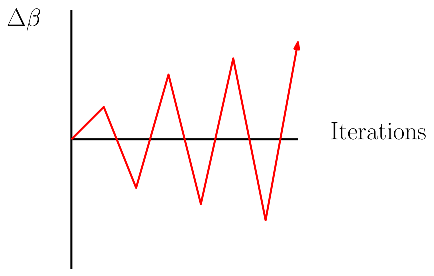
So far we have only used a single thread. Given the large multi-core and multi-CPU systems available, it would be a substantial improvement if we were able to run the algorithm in parallel.
We refer to an update of a $\beta_j$ corresponding to column $X_j$ as a column update. Within a column update, there are no barriers to running in parallel (i.e. parallelising over rows). We can iterate through the elements of the column in parallel using openMP, and calculate the sums with a reduction. The contents of a single column are stored sequentially in memory, which limits the effectiveness of such an approach. The contents of the columns are only read, and not written, in this process, so there is no overhead in maintaining cache coherency. Once a single value has been read on one core, an entire cache-line will be available from its local cache, however. Since these have been read from memory already, there is no advantage to reading them into another core's cache for parallel processing. We could attempt to offset the work of each core, so that each will be working on a separate cache line within the same column of the matrix. Such an approach, however, assumes that the column contains at least $k = \frac{\text{cache line size}}{\text{entry size}} \cdot (\text{num cores})$ entries, which is unlikely. There is also considerable overhead in thread barriers [62], and the work done must be enough to justify this. To solve these problems, and avoid having threads idle when their component of the work is finished, we would need to have several times $k$ entries. In our test set of $n = 10,000$ siRNAs, the mean number of non-zero entries in a column is only 150, or seven compressed 64-bit words. The L1 data cache of our test CPU[2] has a 64 byte cache line size, enough for eight entries. Even with hundreds of thousands of siRNAs, each column could only be expected to be a few cache lines. Running the iterations over columns in parallel, rather than rows, is therefore the focus of our parallelisation attempts.

### 2.4.1 Overlap Error

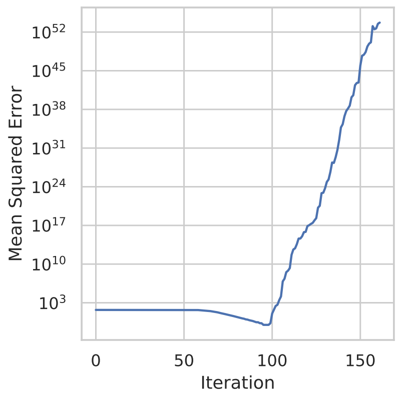
We cannot simply perform column updates in parallel. Each column update both reads and writes $r_i$ values for every non-zero entry in the column. If two columns are updated in parallel, and they both have non-zero entries in a common row, there is a time-of-check to time-of-use problem. An update can occur in an $r_i$ after that value has been read by another thread. While the cached $r_i$ themselves will never be incorrect, as the updates are atomic and always the result of real changes to a beta value, both columns will be updated based on the old value. Both columns are partially responsible for the difference between the current fit, $\sum_{i=1}^{p'} x_{ij}\beta_j$, and observed fitness, $y_i$, of this common row. Each of these updates will attempt to minimise this as much as possible, without taking the other update into account. This can result in overcorrecting for the error in the common row, potentially increasing the overall error. To compensate for the

[2]A Xeon Gold 6244.

increased error, the next update may make an even larger change to $\beta$ (Figure 2.13a). In the worst case, if two or more updates repeatedly overcorrect for each other, this can prevent convergence entirely (Figure 2.13b).



(a)



(b)

Figure 2.13: Effect of repeated overcorrection. (a) Repeated overcorrections lead to increasingly large changes to error. (b) Effect of repeated overcorrection on fit. This example is the result of fitting a 50 $\times$ 20 random binary $\mathbf{X}_2$ matrix, with random $\mathbf{Y}$ values between $-1$ and 1.

## Deriving Overcorrection Error

Simultaneous updates may result in overcorrection, but we can analytically determine exactly how much. Using the definitions from Subsection 2.1.2, ignoring the lasso penalty for the moment, we update a single $\beta_k$ by $\Delta\beta_k$ as follows.

$$\Delta\beta_k = \frac{\sum_{i=1}^{n}(x_{ik}(y_i - r_i))}{S_k}$$

Let us define $\hat{\Delta}\beta_k$ to be the value that $\Delta\beta_k$ would take if, for every $j < k$, the update to $\beta_j$, had already been performed. If we perform all updates strictly sequentially, then $\Delta\beta_k = \hat{\Delta}\beta_k$. Similarly, we define $\overset{\wedge k}{r_i}$ to be the value of $r_i$ after all updates prior to $k$ have been performed.

$$\overset{\wedge k}{r_i} = \sum_{j=1}^{p'}(x_{ij}\beta_j) + \sum_{j=1}^{k-1}(x_{ij}\hat{\Delta}\beta_j)$$

Since the only difference between $\hat{\Delta}\beta_k$ and $\Delta\beta_k$ is the change in $r_i$ caused by previous beta updates, it follows:

$$
\begin{aligned}
\hat{\Delta}\beta_k &= \frac{\sum_{i=1}^{n}(x_{ik}(y_i - \hat{r}_i^{k})}{S_k} \\
&= \frac{\sum_{i=1}^{n}(x_{ik}(y_i - \sum_{j=1}^{p'}(x_{ij}\beta_j) - \sum_{j=1}^{k-1}(x_{ij}\hat{\Delta}\beta_j)))}{S_k} \\
&= \frac{\sum_{i=1}^{n} x_{ik}(y_i - r_i)}{S_k} - \frac{\sum_{i=1}^{n} x_{ik}\sum_{j=1}^{k-1} x_{ij}\hat{\Delta}\beta_j}{S_k} \\
&= \Delta\beta_k - \frac{\sum_{i=1}^{n} x_{ik}\sum_{j=1}^{k-1} x_{ij}\hat{\Delta}\beta_j}{S_k} \\
&= \Delta\beta_k - \frac{\sum_{j=1}^{k-1}\hat{\Delta}\beta_j\sum_{i=1}^{n} x_{ij}x_{ik}}{S_k}
\end{aligned}
$$

Note that $\sum_{i=1}^{n} x_{ij}x_{ik}$ is constant with respect to changes in $\beta$ and $\mathbf{R}$. We can compute these values once after the input has been read, and re-use them in every iteration. If we define the overlap between columns $j$ and $k$ of $\mathbf{X}_2$ to be $\gamma_{jk} = \sum_{i=1}^{n} x_{ij}x_{ik}$, we have the following.

$$\hat{\Delta}\beta_k = \Delta\beta_k - \frac{\sum_{j=1}^{k-1}\gamma_{jk}\hat{\Delta}\beta_j}{S_k}$$

*Remark* 1. $\hat{\Delta}\beta_k \neq \Delta\beta_k$ if and only if $\gamma_{jk} \neq 0$ for some $j < k$.

For $\lambda \neq 0$ we define the soft threshold function $f_\lambda(x)$.

$$f_\lambda(x) = \begin{cases} \min(0, x+\lambda) & \text{for } x < 0 \\ \max(0, x-\lambda) & \text{for } x > 0 \end{cases}$$

We find that the value of $\hat{\Delta}\beta_k$ for $\lambda \neq 0$ is the following, and use this definition in our package.

$$
\begin{aligned}
\hat{\Delta}\beta_{k\lambda} &= f_\lambda(\beta_k + \hat{\Delta}\beta_k) - \beta_k \\
&= f_\lambda(\beta_k + \Delta\beta_k - \frac{\sum_{j=1}^{k-1}\gamma_{jk}\hat{\Delta}\beta_j}{S_k}) - \beta_k
\end{aligned}
$$

We now consider a concrete example of Remark 1. Let each residual $r_i$ be fixed (for the moment), and update first $\beta_1$, then $\beta_2$, the intended sequential update effect is then:

$$\Delta\beta_1 = \frac{\sum_{i|x_{i,1}=1} y_i - r_i}{\sum_{i=1}^{n} x_{i,1}}$$
$$\Delta\beta_2 = \frac{\sum_{i|x_{i,2}=1} y_i - r_i + \gamma_{12}\Delta\beta_1}{\sum_{i=1}^{n} x_{i,2}}$$

When both updates are instead performed at the same time we get:

$$\Delta\beta_1 = \frac{\sum_{i|x_{i,1}=1} y_i - r_i}{\sum_{i=1}^{n} x_{i,1}}$$
$$\Delta\beta_2 = \frac{\sum_{i|x_{i,2}=1} y_i - r_i}{\sum_{i=1}^{n} x_{i,2}}$$

Updating in parallel, the effect of $\beta_2$ is estimated based on the original $\mathbf{R}$, rather than those that account for the changes made to $\beta_1$. We can easily calculate the expected overcorrection in this case, $\gamma_{1,2}\Delta\beta_1$. Note that we are atomically updating the residuals $r_i$ that are affected by both updates. If we fail to do this, overcorrection becomes difficult to predict.
For example, $\beta_1$ and $\beta_2$ correspond to the columns in Figure 2.14a, and suppose these columns of $\mathbf{X}_2$ are chosen for simultaneous updates. We can calculate the changes $\Delta\beta_1$ and $\Delta\beta_2$ in parallel and update the residuals safely, because there are no shared values being updated by both threads. In Figure 2.14b, we find that both the update to $\beta_1$ and the update to $\beta_2$ affect residual $r_3$. While atomic updates to the actual value of $r_3$ will guarantee that we finish with the value $r_3 + \Delta\beta_1 + \Delta\beta_2$, we have not taken the changed value of $r_3$ into account when calculating $\Delta\beta_2$. The correct update would have been $r_3 + \Delta\beta_1 + \Delta\beta_2 - \frac{\Delta\beta_1}{2}$

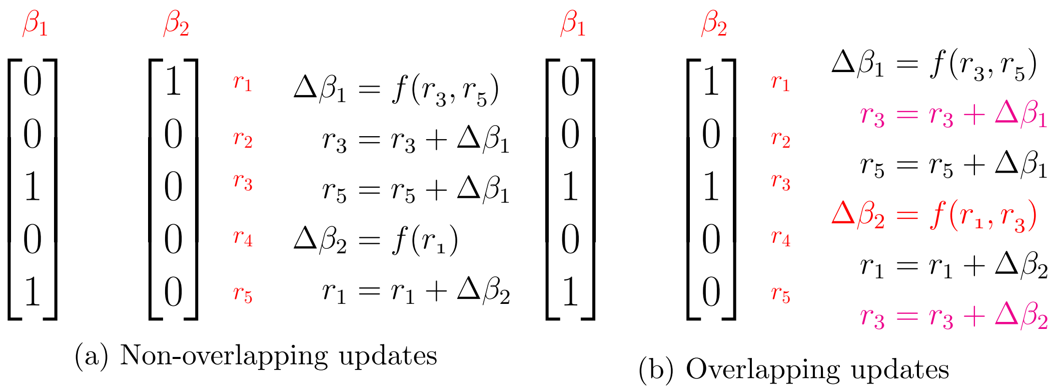

$\beta_1$ $\beta_2$
$\begin{bmatrix}0\\0\\1\\0\\1\end{bmatrix}$ $\begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix}$
$r_1$ $r_2$ $r_3$ $r_4$ $r_5$
$\Delta\beta_1 = f(r_3, r_5)$
$r_3 = r_3 + \Delta\beta_1$
$r_5 = r_5 + \Delta\beta_1$
$\Delta\beta_2 = f(r_1)$
$r_1 = r_1 + \Delta\beta_2$
(a) Non-overlapping updates
$\beta_1$ $\beta_2$
$\begin{bmatrix}0\\0\\1\\0\\1\end{bmatrix}$ $\begin{bmatrix}1\\0\\1\\0\\0\end{bmatrix}$
$r_1$ $r_2$ $r_3$ $r_4$ $r_5$
$\Delta\beta_1 = f(r_3, r_5)$
$r_3 = r_3 + \Delta\beta_1$
$r_5 = r_5 + \Delta\beta_1$
$\Delta\beta_2 = f(r_1, r_3)$
$r_1 = r_1 + \Delta\beta_2$
$r_3 = r_3 + \Delta\beta_2$
(b) Overlapping updates

Figure 2.14:

Given that we can safely perform updates for columns that have no overlap, and we can explicitly compensate for the error of sets of columns, we investigate three approaches for parallelisation: compensating for the error of pre-determined sets (Subsection 2.4.2), simultaneously updating non-overlapping sets (Subsection 2.4.3), and randomly updating shuffled columns (Subsection 2.4.4).

### 2.4.2 Explicit error compensation

To update $\beta_a$ and $\beta_b$ at the same time, we need to subtract the overcorrection from one of them (arbitrarily chosen) afterwards. The final value is then the same as if we had updated the values sequentially. Subtracting the overcorrection from $\Delta\beta_a$ we update $\beta_a$ as follows:

$$\beta_a = \beta_a + f_\lambda(\beta_b + \Delta\beta_k - \frac{\gamma_{ab}\hat{\Delta}\beta_b}{S_k}) - \beta_a$$

Similarly, we can update any subset of the beta values $\{\beta_1, \ldots, \beta_{p'}\}$ simultaneously, as long as we account for overcorrection in each update. For every $\beta_k$ in the subset $\{\beta_1, \ldots, \beta_l\}$, we make the following correction:

$$\beta_k = \beta_k + f_\lambda(\beta_k + \Delta\beta_k - \frac{\sum_{j=1}^{k-1}\gamma_{jk}\hat{\Delta}\beta_j}{S_k}) - \beta_k$$

This method has been implemented in the `error_comp` branch of our repository[3], and is marginally faster than sequential updating with the right parameters. It does not scale well enough that we can recommend its use, however.

To understand the scalability of this approach, we begin by noting that the time taken to correct $C$ simultaneous beta updates is on the order of $C^2$. This is because each update $0 \le i \le C$ requires reading the $i-1$ previous corrected values, resulting in $\frac{(i-1)(i-2)}{2}$ reads. If we were to attempt to update the entire interaction matrix in parallel, followed by correcting errors, there would be on the order of $p'^2$ corrections. Since a sequential iteration requires only $p'\mu$ steps, where $\mu$ is the mean number of non-zero entries per column, we would spend more time on corrections than updates. Even if corrections were run in parallel, this would be slower than sequential updates.

We do not, however, have to update the entire matrix at once. If we restrict ourselves to updating small sets, where 'small' is some function of the number of threads we are able to effectively use, the problem becomes tractable. Performing $C$ parallel updates, where $C$ is some constant multiple of the number of available threads, we have in total $\frac{p'}{C}$ sets to update, resulting in $\frac{p'\mu}{C} + p'C$ update operations on the main thread.[4] For $\frac{C^2}{C-1} < \mu$ this is an improvement, even with single-threaded correction.[5]

*Remark* 2. Ahmdahl's Law implies a best-case improvement of:

$$\frac{p'\mu}{p'(\frac{\mu}{C} + C)}$$

[3]`https://github.com/bioDS/lasso_testing/tree/error_comp`

[4]Assuming that the column updates are done in parallel, and the overcorrection adjustments are done on the main thread.

[5]Note that $\frac{C^2}{C-1} \approx C$ for sufficiently large C

### Parallel correction does not scale

To improve upon Remark 2, we have to correct beta updates in parallel. This requires calculating $\hat{\Delta}\beta_k$ without using any previously corrected values, $\hat{\Delta}\beta_j$ for $j < k$. Substituting $\hat{\Delta}\beta_j$, a corrected update $\hat{\Delta}\beta_k$ then becomes the following:

$$\hat{\Delta}\beta_k = \Delta\beta_k - \frac{\sum_{j=1}^{k-1} \gamma_{jk}\Delta\beta_j - \frac{\sum_{j'=1}^{j-1}\gamma_{j'j}\Delta\beta_{j'} - \Gamma}{S_j}}{S_k}$$

Where $\Gamma$ is a nested sum containing a further $(k-4)!$ additions. Even for very small sets, computing this is not feasible. We must therefore fix overcorrection sequentially, and accept the limit in Remark 2.

### OpenMP barrier overhead requires large $C$

To achieve the improvement in Remark 2, we would need a set of eight columns, updated on eight CPUs, to finish eight times faster. This is in practice not the case. We demonstrate this by running a single iteration on the same test set with varying parallel update set sizes $C$. Using sets of size $m$ times the number of available cores we gradually increase $m$ and measure the time taken to perform all $p'$ column updates, without compensating for overcorrection. As we can see in Figure 2.15, we require blocks of 64 to 128 times as many columns as there are cores to achieve speed-ups close to the theoretical limit, and at least eight times as many for any significant improvement.

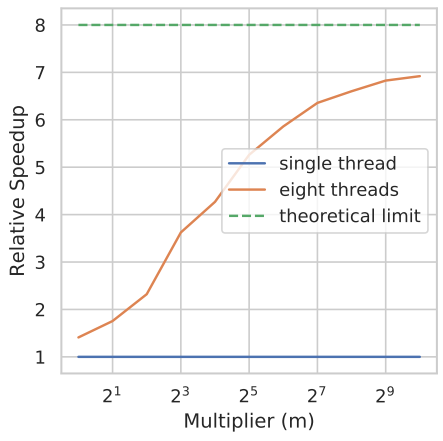


Figure 2.15: Set size multiplier effect on parallelisation speed-up. Sets contain (number of cores) $\cdot$ $C$ columns.

We find that for small block sizes, the majority of the time all threads are handling an openMP barrier at the end of a parallelised loop.[6] There are two likely causes. First, when running a small number of column updates in parallel, some of these will often finish before others. If there are no further columns in the set, the thread will then

[6]Note that it is not the case that one thread is still updating while the others are waiting.

have to wait for the set to finish updating. Secondly, updating columns with entries in nearby rows will mean accessing nearby memory locations to update the residuals. Since these are both being read and written to, maintaining cache coherency across CPU cores is likely to affect performance. The barrier at the end of each parallel section may also take a significant amount of CPU time, as it requires some communication between threads.

Memory access is also less efficient with multiple threads. When a single thread performs a sequence of column updates, these columns are stored in memory sequentially, and each column may be as little as a single word. The core will read in an entire cache-line at once, and this may contain part of the following column, if not the following several. In this case, additional updates for these columns may be performed without any extra memory reads. If, on the other hand, we have eight columns shared between eight threads, they will each still need to read in these values. In our case the eight cores have a shared L3 cache, individual L1 and L2 caches, and 64 byte cache lines. Since one Simple-8b word is 64 bits, we have eight such words in a single cache line. In the worst case, this is eight separate columns. Supposing this is the case, a single read from memory will bring all eight columns into the shared L3 cache. It will then take eight separate reads into various L1 caches, for eight separate cores to perform updates for these columns, whereas a single core would require only one read from L3. With larger data sets, and hence larger columns, this becomes less of a problem. The fact remains though, that cache use is significantly better when each core updates several sequential columns.

All of these issues are mitigated by increasing the size of sets, and updating several sequential columns on each thread. Figure 2.15 suggests sets should be at least sixteen times the number of available threads in size. This increases the time required to compensate for overcorrection.

### There is no effective value of $C$

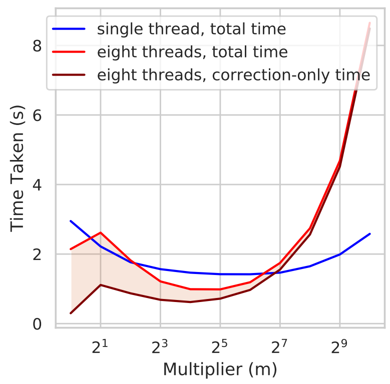


Figure 2.16: Total time taken (in seconds) for various block size multipliers. Time spent in parallel section is highlighted. Single-threaded time is included for comparison.

The results in Section 2.4.2 require us to perform error correction sequentially, i.e. on a single core. As we saw in Remark 2, for sufficiently small sets this is not a serious limitation. However Section 2.4.2 also suggests that we may need large sets.
We first note that we can't significantly reduce the time taken in the error correction step. Updating **R** and calculating corrections take approximately half the time each. While updating **R** can be parallelised, there is so little work here that the thread barriers again result in worse performance. In the overlap matrix, containing the overlap between columns in the set ($\gamma_{ij}$ for columns $i$, $j$), around 50% of the entries are non-zero. Removing these or using a sparse compressed matrix to store overlap is therefore unlikely to be a significant improvement.
Figure 2.16 shows the amount of time spent in error correction increases quadratically with the block size. At a multiplier of 256 this overtakes the entire update time on a single core, and we see that the multi-threaded implementation becomes slower than the single-threaded. The best multi-threaded result we see is at a multiplier of thirty-two, where the parallel version achieves a 44% improvement in iterations per second. Even with improvements to the error correction routine, it is unlikely that this approach to parallelisation can achieve more than double the performance of the sequential version. We therefore arrive at the conclusion:

*Remark* 3. Parallel updates of sets, followed by error correction on those sets, is not a feasible approach for parallelisation.

### 2.4.3 Limiting overlap

While we cannot compensate for overcorrection after the fact, Figure 2.15 nonetheless suggests that there is some hope for performing a block of updates in parallel. Rather than allowing these blocks to be arbitrary, we now consider restricting updates to blocks of non-overlapping columns of $\mathbf{X}_2$. Again, we first divide the matrix into sets of columns for which we can perform updates simultaneously, then update these sets one at a time. Here, these sets will be collections of columns that either do not overlap, or overlap very little.

#### Sets of no overlap

Since the columns of $\mathbf{X}$ are relatively sparse, and the columns of $\mathbf{X}_2$ particularly so, it is plausible that we could find sets of a few non-overlapping columns purely by chance. In our test set of $1,000$ columns and $10,000$ rows the mean number of non-zero entries in a column is 0.58%. In this case, we would expect the fraction of entries in common between two randomly chosen columns to be $(0.58\%)^2$, or 0.34 entries per column. If we choose two random columns we can reasonably expect them not to overlap. Extending this we find sets of non-overlapping columns using the following method.
We start by randomly shuffling the columns. We then compare every second column with its neighbour, recording a set of two non-overlapping columns where possible (Figure 2.17b). Smaller sets are then repeatedly merged to form larger sets, where no overlap exists between columns (Algorithm 2). The algorithm described in Algorithm 1 is then run, updating beta in parallel for each set of non overlapping columns.

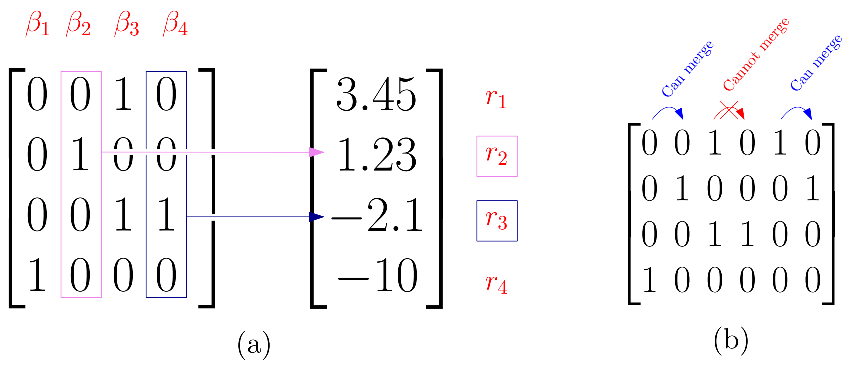


Figure 2.17: (a) Completely non-overlapping updates affect different residual values and can be done safely in parallel. (b) Attempting to merge neighbouring columns.

**Algorithm 2:** Mergesets Algorithm

```
Input: $X \in \{0,1\}^{n \times p}$
Result: Sets of non-overlapping columns
Initialise all columns as one element sets
Merge neighbouring sets where no overlap exists
Place sets in appropriate bin (bin 1: sets of size 1, bin 2: sets of size 2, ...)
for small_bin in bins do
    for large_bin in bins s.t. large_bin > small_bin do
        n ← min(sizeof(small_bin), sizeof(large_bin));
        for offset in $0, \ldots, 50$ do
            Attempt to merge the first n elements of small_bin with the first n
              elements of large_bin, starting at offset.
        end
    end
    Attempt to merge the first half of small_bin with the second half.
end
```

### Performance of Overlap Method

We compare run time on one and four cores, using simulated sets of 1000 siRNAs and 100 genes (i.e. pairwise interactions on a $1000 \times 100$ matrix) in Figure 2.18. Once sets of non-overlapping columns have been found, updating non-overlapping sets in parallel improves run time compared to the single-threaded version (Figure 2.18b).

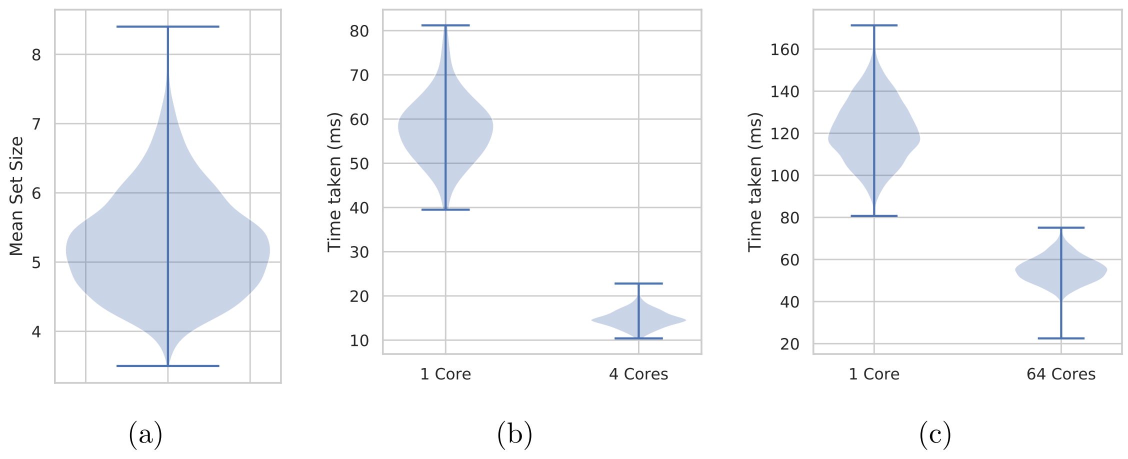


Figure 2.18: (a) Mean non-overlapping set sizes. (b) Run time after finding mergesets on four vs one core. (c) Run time after finding mergesets on 64 vs one core. Note that the 64 core system has considerably slower memory than the four core system.

As the size of input data increases, however, finding sets with absolutely no overlap becomes more difficult. Since, as explained in Subsection 2.4.2, efficiently using more threads requires larger set sizes than those in Figure 2.18a, we aim to improve on these. Running more comparisons with our current implementation is not feasible, finding sets already takes as long as the regression step. Instead, we relax the criteria from no overlap to very little overlap.

## Partial Overlap

We can find much larger sets of simultaneously updateable columns if we allow a small amount of overlap between these columns. While this does allow some error to be introduced in the calculation of beta updates, we expect that by limiting the overlap this error will remain small, and not result in the drastic overcorrection seen in Figure 2.13.

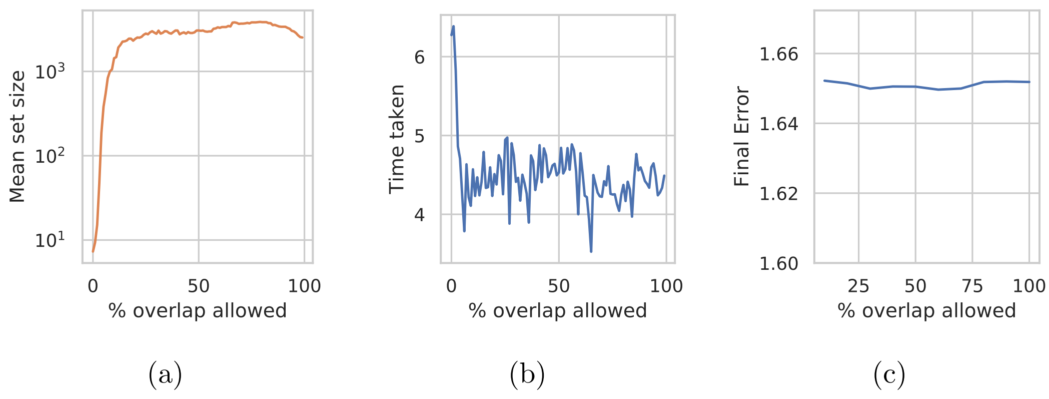


Figure 2.19: (a) Mean set size, (b) run time, and (c) final mean squared error on 64 cores as allowed overlap increases from 1% to 100%.

In small test sets ($n = 1000,\ p = 100$), increasing the available overlap significantly increases the found set size (Figure 2.19a), also improving the run time. Interestingly, increasing the allowed overlap to 100% does not harm the run time (Figure 2.19b) or the mean squared error of the final fit (Figure 2.19c). We would expect both of these to suffer when the columns significantly overlap, as either further iterations are required to correct for the introduced error, or overcorrection error is allowed to remain. It appears that even allowing 100% overlap, there is a negligible amount of overcorrection occurring.

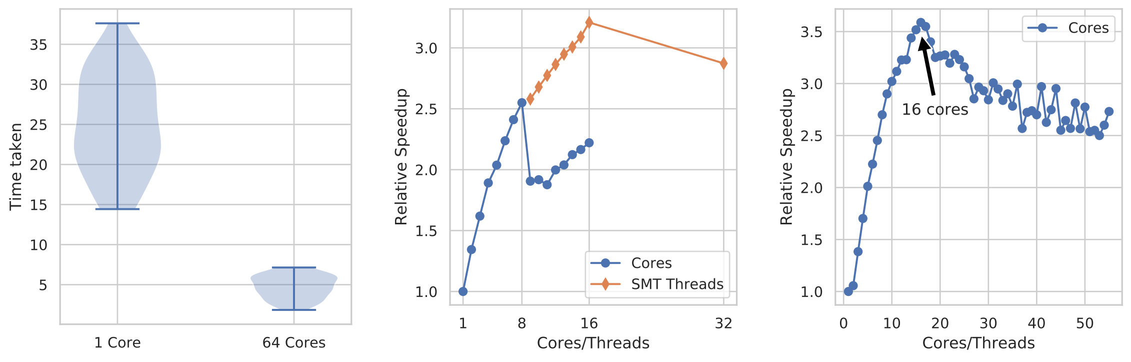
### 2.4.4 Random Overlap

Performance with 100% overlap allowed suggests that we can avoid finding simultaneously updateable sets entirely, relying instead on the relatively small overlap between randomly chosen columns. This was shown to work by Bradley et al. [8] for their lasso implementation, with up to $\frac{p}{\rho} + 1$ parallel updates, where $\rho$ is the spectral radius of $\mathbf{X}^T\mathbf{X}$, so long as the columns being updated simultaneously were chosen at random. In our case, using the matrix of pairwise interactions, this allows $\frac{p(p+1)}{\rho} + 1$ parallel updates, where $\rho$ is the magnitude of the largest eigenvalue of $\mathbf{X}_2^T\mathbf{X}_2$. In our smallest test case ($n = 1,000,\ p = 100$), this would allow 222 simultaneous updates. This number increases with larger input data. Given the significant cost of finding sets, this is the approach we take from here on. To ensure the columns being updated simultaneously are always chosen at random, we shuffle the column order in every iteration of the regression. Combining this with the beta updates from Equation 2.4, and the lambda sequence from Section 2.2, we have our final lasso algorithm, Algorithm 3.

**Algorithm 3:** Shuffled Lasso Algorithm

```
Input: $X \in \{0,1\}^{n \times p}$, $Y \in \mathbb{R}^n$, error_cutoff
Result: beta values
for lambda in lambda sequence do
    while $\frac{old\ error}{new\ error}$ > error_cutoff do
        for column in columns do
            $\beta_{\text{column}} \leftarrow \beta_{\text{column}} + \Delta\beta_k$
        end
        error $\leftarrow \sum_{i=1}^{n} y_i - \sum_{j=1}^{p'} x_{ij}\beta_j$
    end
    if adaptive calibration conditions met OR maximum non-zero betas found
     then
        Stop after current lambda
    end
    else
        $\lambda \leftarrow \lambda \times 0.9$
    end
end
```

**Final parallel vs sequential performance**



(a) (b) (c)

Figure 2.20: Relative speedup as the number of cores used increases. (a), (b) Running on a dual 8 core/16 thread NUMA system. (c) Running on an eight CPU NUMA system without SMT. Each CPU has 8 cores, and memory is divided among four packages of two CPUs, where memory is shared within packages.

Running this shuffled version (Algorithm 3) on a 16-core NUMA system, we see a reasonable speedup using up to eight cores. The ninth core, however, sees a significant drop in performance (Figure 2.20b). Given that this is the first core of the second CPU, this can be explained by the need to keep a second L3 cache coherent and significantly slower memory access from the second NUMA node. We see continued improvement when increasing the number of cores on the second node, but never enough to outperform using only the first node. Using multiple SMT threads also improves performance, and the best speedup we are able to obtain (3.2 times) is using all 16 threads on a single NUMA node. Increasing SMT threads beyond a single NUMA node also harms performance. We see a similar situation in Figure 2.20c, where performance increases up to the number of cores in the first package. Although this is actually two NUMA nodes, they share the same memory.

## 2.5 Limited Interaction Neighbourhoods

When searching for strong negative interactions within a sequence of genes, it may be acceptable to limit the search to pairs that are relatively close on the gene sequence. In a study of epistatic interactions in yeast by Puchta et al. [72] the strength of negative interactions decreases as distance between gene positions on the sequence increases. In fact, the median distance between pairs in the hundred strongest interactions was only eighteen nucleotides.

Limiting interactions to those within some distance $d$ drastically reduces both the time and space requirements. Instead of $\Theta(p^2n)$, the size of the interaction matrix becomes $\Theta(pdn)$. Similarly, an iteration of Algorithm 3 would require only $\Theta(pdn)$ operations. For $d << p$ this is a significant reduction. Limiting the interaction search distance to 100 positions, we could process a set of $30,000$ genes and $200,000$ siRNAs using approximately $4GB$ of memory, assuming a comparable density of interactions to our testing data. Such a search could be performed directly on a laptop, without requiring access to a large server. The biological implications of this restriction should be carefully considered before its use, however.

## 2.6 Discussion

### 2.6.1 Performance Comparison

We compare our method to `glinternet` [53], the method demonstrated to effectively find interactions in small data sets in Chapter 1. Since we achieved the best results only using `glinternet` for variable-selection, then fitting the non-zero beta values with ordinary least squares regression, we do the same here. We use our parallel lasso in the same way and restrict to the first $2,000$ non-zero beta values, rather than using adaptive calibration, which returns too few columns for the OLS regression step.
Testing with the same data as in Chapter 1, our method is able to identify significantly more correct interactions than `glinternet` (Figure 2.21). Precision is largely comparable, with a few outliers in which we see significantly more false positives with our method (Figure 2.21a). The run time is orders of magnitude faster than `glinternet`, typically taking 20 to 30 seconds rather than several hours (Table 2.1, Figure 2.21c). To test the scalability of our implementation, we also run it with the same $2,000$ effect limit on a much larger data set. With $p \approx 27,000, n \approx 30,000$, using 16 SMT threads on a single eight core CPU, we propose 97 main effects and 236 interactions in one and a half hours.

| $\mathbf{X}$ matrix size | `glinternet` time (s) | compressed lasso time (s) |
|---|---|---|
| $n = 100,\ p = 1,000$ | 178 | 2.00 |
| $n = 1,000,\ p = 10,000$ | 4807 | 27.6 |
| $n = 30,000,\ p = 27,000$ | ... | 5889 |

Table 2.1: Runtime comparison between our method and `glinternet`.

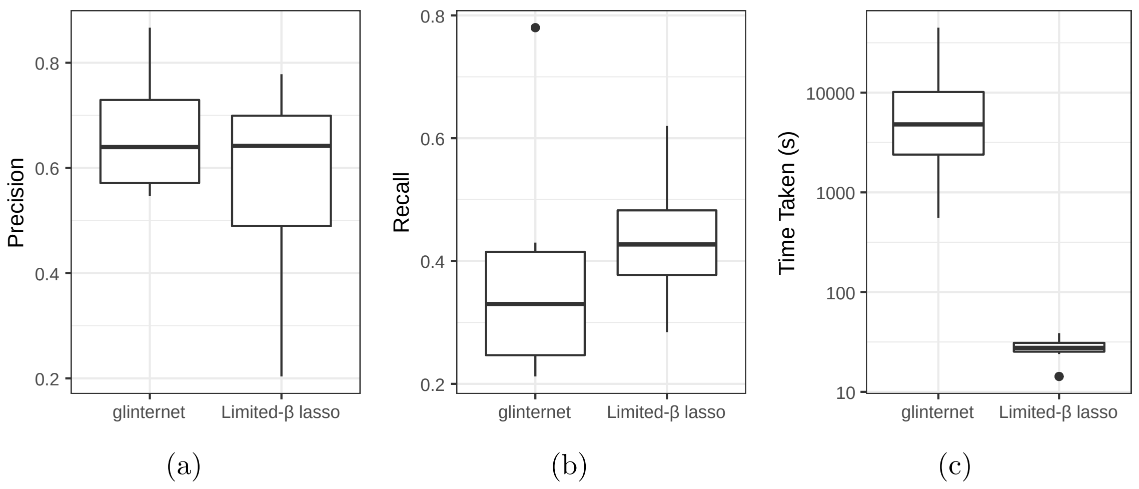


Figure 2.21: Searching for interactions with `glinternet` vs. our shuffled compressed lasso, using $p = 1,000, n = 10,000$ data from Chapter 1. (a) Precision. (b) Recall. (c) Time taken.

### 2.6.2 Limitations and Future Work

If the original $\mathbf{X}$ matrix is sparse, and the pairwise interaction matrix $\mathbf{X}_2$ is very sparse, we would expect three-way interaction columns of an $\mathbf{X}_3$ matrix to be *extremely* sparse. If there are few enough non-zeros in such a matrix, it may be possible to extend our method beyond pairwise interactions. While there would be $p^3$ columns in a three-way interaction matrix, if the vast majority contain only zeros we may still be able to store it. The indices of non-zero three-way interaction columns could themselves be stored in a compressed list of offsets. Any column whose index is not in this list could then be presumed to be zero and left out of beta updates. If the number of non-zero columns does not drastically increase, this may be feasible.
Accessing randomly shuffled columns on NUMA systems, when columns are moved across CPU memory boundaries, is very inefficient. It is only necessary to share changes to $\mathbf{R}$ between nodes every iteration, and it would be significantly more efficient to divide the matrix between nodes, and shuffle only the component local to each node at each iteration. The significantly higher memory throughput available to such a system could then be used efficiently, rather than slowing the program down. This will not work in the extreme case where every node has only a single column, but we expect a significant number of nodes could be efficiently used for large (exome-scale) input. Shuffling the columns every iteration is currently done on a single thread, and fast algorithms to do this in parallel exist[47]. Running this in parallel would avoid any part of the algorithm being single threaded, leaving no limit imposed by Ahmdal's Law.

## 2.7 Conclusion

Beginning with an efficient serial algorithm for lasso regression, we investigated a variety of methods for parallelisation. As long as there is no overlap between columns, we can perform parallel updates. We investigated guaranteeing this restriction by finding sets of non-overlapping columns, limiting overcorrection by using sets of mostly non-overlapping columns, and relying on the sparsity of the matrix by randomly shuffling columns. While all three methods improved performance over the serial algorithm, random shuffling requires no preprocessing of the interaction matrix, and has the best performance overall. By using an efficient representation of the interaction matrix this can be done on a sufficiently large shared memory system, without requiring a GPU, in a matter of hours. If interactions are restricted to those within 100 positions of each other, we could search $n = 200,000$ siRNAs and $p = 30,000$ genes using only 4GB of memory. Comparing our shuffled implementation to `glinternet`, we are able to achieve marginally better prediction of genetic interactions in several orders of magnitude less time. Given a sufficiently sparse matrix, we are able to perform lasso-regularised linear regression on all pairwise interactions between human protein-coding genes.

# Chapter 3

# Discovering Functional Modules from Pairwise Interactions and Expression Data

In Chapter 2 we presented a method improving on the state of the art methods for identifying pairwise interactions, using a second-order Taylor approximation of interactions. This method assumed that interactions beyond two genes were unimportant, and that fitness is a linear combination of gene effects. While this is sufficient for identifying epistasis, it is an extreme simplification of genetic interactions. In this chapter we discuss a more general model, in which genes interact in functional modules, and consider how our earlier results may best be integrated into such a model.

To better organise and understand the complex data that arises, a variety of biological processes are modelled as networks [4]. Common examples include protein interactions [84], metabolic pathways [23], and transcription regulation [97]. Many of these networks are believed to contain functional modules, groups whose members are involved in performing a common task, where there are many strong connections within the group, and few connections outside [81]. Identifying these modules is a common goal in systems biology [39, 54, 55]. The meaning of edges in these networks varies significantly, edges can for example represent known physical interactions [66] or association in text-mining [67]. Nonetheless, these networks are often believed to contain the same modules, and many recent efforts have focused on combining several sets of data [20, 7, 68].

We propose a clustering method to do so that combines a network of pairwise interactions associated with genes, protein-protein interactions or epistatic interactions for example, with a normalised set of expression data. Our method uses the expression data to add edges to the pairwise interaction graph, clusters the result using the SLPA algorithm [101], and selecting the best clusters using an extension of modularity density [12]. In our simulations, using the expression data to add edges to the pairwise interaction graph, we are able to improve the accuracy of detected clusters over clustering the pairwise interaction graph alone. We evaluate our method's ability to provide a broad overview of gene activity by comparing cluster activity between different tissues. Our simulation and clustering package is developed in Rust, and will be available at

`https://github.com/biods/gene_clustering`.
We integrate pairwise interactions of the form discussed in chapters 1 and 2 with gene expression data to discover functional modules of genes. Pairwise interactions may be from any source, not only our earlier regression model (Chapter 1). In particular, we intend our method to also work with protein-protein interactions, as several databases of known and suspected protein interactions already exist [85, 43]. Our aim is to combine a graph of such interactions with experimental measurements of gene expression to produce a more accurate clustering than we could achieve with the pairwise interaction graph alone. Doing so, we aim to provide a more interpretable overview of gene activity in terms of clusters.
We investigate clustering a combined graph in which weighted co-expression edges are added to pairwise interactions, and conclude that such a method requires filtering out many of the co-expression edges beforehand. Once this has been done, we find that the combination significantly improves the accuracy of proposed clusters in our simulations. We first cluster the genes using both the expression-correlation and protein-protein interaction (PPI) graphs, scoring these clusters with modularity to identify those that fit best. Secondly, we consider filtering out the majority of the co-expression edges, leaving only those that represent significant correlation. Finally we present the expression data in terms of these clusters. There are a number of common difficulties that we do not attempt to address here, and we do not include these in our simulations. Protein-protein interactions, for example, are known to often contain false positives [59], as well as missing many previously known interactions [34]. Furthermore gene expression is difficult to reliably measure [75], and expression measurements require normalisation [33]. We assume that gene expression measurements have been normalised, and that measurement errors have been accounted for outside of our program. While we recognise that data in this field often contains errors, addressing this is beyond the scope of this project.

## 3.1 Background

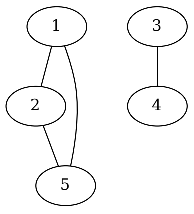
Graph clustering, identifying densely connected subgraphs, is a well studied problem, and many general techniques exist [76]. Recently, attention has focused on extracting information from biological networks, including using protein interactions to discover protein complexes [99] [96] and functional modules [95] [40]. Clustering gene expression data can be used to identify biologically significant groups [41]. To improve accuracy, multiple sources of data have been combined in a number of cases. Keretsu and Sarmah [45] for example used gene expression data to add weights to protein interaction graphs. Multiple biological graphs may be clustered simultaneously to discover their common clusters as in Ma, Sun, and Zhang [56]. Moreover, expression data can be added to filter out the best clusters from a protein interaction graph [103]. Recently, protein interactions and gene expression have also been combined using multiplex clustering techniques. Liang et al. [52] used gene expression and text-mining to produce two differently weighted versions of the PPI graph, and R. and Nazeer [73] combined a variety of graphs to form a weighted consensus. Both of these methods focus on reducing the clusters to those that all data sets agree on.

We combine an arbitrary network of pairwise interactions (of which protein interactions are a typical example) with gene expression data in a way that will not only find clusters present in all sets, but also allow clusters strongly supported by only one set to be found. To that end, we focus on adding edges based on the expression data, rather than weighting existing ones or filtering out clusters.

## 3.2 Input Data

The two inputs we consider are a pairwise interaction network, and a set of gene expression data, across various tissues (Figure 3.1b). We model the pairwise interactions as undirected edges on a graph of genes. As typical examples of a pairwise network we consider protein-protein interactions and epistatic interactions. A variety of methods exist for constructing protein interaction networks, ranging from measuring protein complexes with mass spectrometry [28] to text-mining [67]. Regardless of the method used to find protein interactions, we can produce a list of pairs of genes that are either known or suspected to interact. We construct an unweighted undirected graph by placing an edge between every pair of genes that encode the interacting proteins (Figure 3.1a). If we were to use epistatic interactions instead, we would place an edge between any pair of genes believed to interact. Gene expression is typically measured by taking a gene product, for example a produced RNA, and using this as a proxy of the level of activity [61, 94]. To turn this into a usable co-expression graph, we take the Pearson correlation, across all sets of measurements, of every pair of genes, and make these the edge weights in a complete weighted graph. The pairwise interaction and the co-expression graphs have significant topological differences, making a single clustering technique difficult to apply to both. The pairwise interaction graph is an unweighted graph with clusters defined by its topology. The co-expression correlation graph on the other hand is a complete weighted graph, in which clusters are defined by the shared weights.
To provide a known true set of clusters for evaluating our method, we simulate both the pairwise network and the co-expression data. Simulations also allow us to precisely describe the noise in both the network and expression data.

```
    gene1 -
gene2
    gene1 -
gene5
    gene2 -
gene5
    gene3 -
gene4
```



(a)

| | Tissue | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| gene 1 | 0.99 | 0.12 | 0.76 | 0.21 | 0.33 | 0.66 |
| gene 2 | 0.92 | 0.09 | 0.55 | 0.15 | 0.23 | 0.55 |
| gene 3 | 0.21 | 0.54 | 0.22 | 0.92 | 0.67 | 0.89 |
| gene 4 | 0.22 | 0.65 | 0.15 | 0.78 | 0.56 | 0.76 |
| gene 5 | 0.96 | 0.11 | 0.81 | 0.18 | 0.21 | 0.33 |

(b)

Figure 3.1: (a) Example protein-protein interaction network. (b) Example expression Data.

### 3.2.1 Simulation

We simulate data with known clusters by first defining the clusters, and then producing two sets of data. First, a pairwise interaction network in which the clusters are more densely connected than the rest of the graph. Second, a set of simulated expression measurements, in which genes present in the same cluster have correlated expression. We use a stochastic block model to simulate interaction networks with known clusters. This model has also been used as an approximation for protein interaction networks [20]. The parameters used in our simulation are chosen arbitrarily. While it would be more informative to test our method using data simulated with a variety of parameters, and identifying those for which our method works well, this is beyond the scope of our project. A more thorough evaluation of different networks would be a worthwhile subject for future investigation.

#### Pairwise Network

Protein interaction graphs commonly exhibit the small-world effect [4], in which any two vertices can be connected with a relatively short path. They have also been observed to follow a power law [37], and be scale-free [4], although not in all cases [87]. The presence of hub proteins, connected to a large number of other proteins, is also common [69]. Simulation of realistic protein interactions is therefore a challenging task, and as we do not intend our method to be restricted solely to protein interactions, it is not one that we attempt to undertake here.
Instead, a network of $n$ genes is generated using Algorithms 4 and 5. First, we take a graph with $n$ vertices and no edges. The vertices are then randomly assigned to clusters. Edges are added within clusters at random with a certain probability, and between clusters with a lower probability. Finally, several edges between clusters are chosen at random, and vertices at both ends are merged into the cluster at the other end, producing overlapping clusters.

**Algorithm 4:** Generate Partition Algorithm

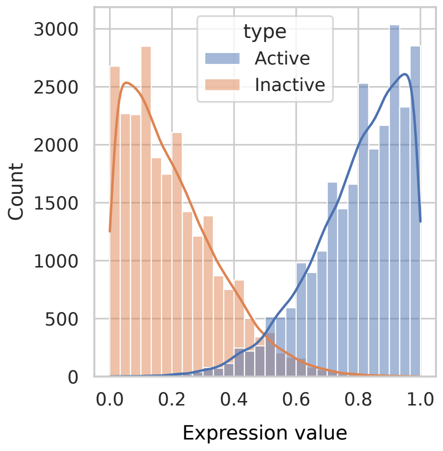
```
Input: Mean set size, number of vertices $n$
Result: Graph $G = (V, E)$, clusters $C = \{c_1, \dots, c_j \subset V^n\}$, expression
        $K_v \in \mathbb{R}^n, K_c \in \mathbb{R}^j$
$V \leftarrow \emptyset$;
$V_{unassigned} \leftarrow \{1, \dots, n\}$;
$E \leftarrow \emptyset$;
while $|V_{unassigned}| > 0$ do
    new cluster size $\leftarrow min(Binomial(10, 0.7))$;
    new cluster $c \leftarrow \{v_1, \dots, v_{size}\}$ at random from $V_{unassigned}$;
    $C \leftarrow C \cup \{c\}$;
    $V \leftarrow V \cup \{c\}$;
    $V_{unassigned} \leftarrow V \setminus \{c\}$;
    $k_c \leftarrow 0$ or 1 at random;
end
for $v \in V$ do
    connect $v$ to $Binomial(10, 0.1)$ other vertices at random;
    for $v'$ in the same cluster as $v$ do
        connect $v, v'$ with probability 0.3;
    end
end
$G \leftarrow (V, E)$;
$K \leftarrow \{k_1, \dots, k_j\}$;
```

**Algorithm 5:** Generate Melted Partition Algorithm

```
Input: Graph $G = (V, E)$, clusters $C = \{c_1, \dots, c_k : \forall i \; c_i \subseteq V\}$
Result: overlapping clusters $C'$
for $v \in V$ do
    $neighbours \leftarrow \{c \in C$ such that $\exists v' \in c : (v, v\prime) \in E, v \notin c\}$;
    foreach neighbour $c$ in neighbours do
        with a 20% chance: $c \leftarrow c \cup \{v\}$;
    end
end
$C' \leftarrow \{c_1, \dots, c_k\}$;
```

### Expression

Once these overlapping clusters have been produced, we simulate expression sets for $t$ separate tissues. To do this, we set each cluster to be either active or inhibited at random with equal probability. We then iterate through each of the $t$ separate tissues, randomly changing the state of each cluster at the current tissue with a low probability. In each tissue, the expression value for each gene is then sampled from one of two distributions, depending on the state of its cluster (Figure 3.2). For genes present in multiple clusters this expression is additive, on the assumption that genes that are involved in multiple processes will be expressed more than those that are not. This produces a table of expression data where changes in clusters are correlated across different tissues (such as Figure 3.1b).



$$f(x) = \begin{cases} 1 - |\mathcal{N}(0, 0.25)| & \text{for } x = 1 \\ |\mathcal{N}(0, 0.25)| & \text{for } x = 0 \end{cases}$$

Where we retry if $f(x) < 0$ or $f(x) > 1$.

Figure 3.2: Observed distribution of gene expression values given cluster state on the left. Simulated distributions on the right, where $x$ is the cluster activity.

## 3.3 Clustering Method

From a set of pairwise interactions and several sets of gene expression data, we first calculate the Pearson correlation between each gene in the expression data sets. Using this we create a weighted undirected graph with edges weighted by the Pearson correlation for each pair of genes.
Given this and the pairwise interaction graph, there are a number of clustering methods that could be used. We use the SLPA algorithm [101], originally intended for clustering social networks. This is a recent method with a remarkably short run time. While the algorithm was not developed specifically for biological data, it performs well compared to state of the art alternatives in general graph clustering problems [30], and can be parallelised [46]. Given the short run time and potential for parallelisation, we expect this method to perform well with large data sets, allowing genome-wide clustering. We allow clusters to overlap, such that vertices may be in multiple clusters. Vertices may be present in different clusters to varying degrees, and we say that vertex $v$ is in cluster $c$ with weight $a_{v,c}$. We use the following definitions in our benchmarking:

$$\begin{aligned}
C &= \text{set of found clusters} \\
C_T &= \text{set of true clusters} \\
F_c &= \sum_{v \in c} \sum_{v' \in c} \sum_{\substack{c' \in C_T: \\ v,v' \in c'}} a_{v,c'} \cdot a_{v',c'} \quad \text{correctly found shared weight in cluster } c \\
T_c &= \sum_{v \in c} \sum_{v' \in c} \sum_{\substack{c' \in C: \\ v,v' \in c'}} a_{v,c'} \cdot a_{v',c'} \quad \text{total shared weight for all genes in cluster } c \\
E_c &= \sum_{v \in c} \sum_{v' \in V} \sum_{\substack{c' \in C_T: \\ v,v' \in c'}} a_{v,c'} \cdot a_{v',c'} \quad \text{number of pairs of genes in cluster } c \\
precision &= \frac{F_c}{T_c} \\
recall &= \frac{F_c}{E_c} \\
F1 &= \frac{2 \cdot precision \cdot recall}{precision + recall}
\end{aligned}$$

## SLPA

The Speaker-listener Label Propagation Algorithm (SLPA) [101] is as follows.

**Algorithm 6:** SLPA Algorithm

```
Input: $G = (V, E)$
Result: List of clusters with membership scores for each node
Each node has a memory of labels that it has heard.
Initially this will include only the unique label of that node.
for $iter \in (1, \ldots, 20)$ do
    foreach node do
        foreach neighbour of node do
            neighbour chooses a label from its memory.
            This is added to the set of spoken labels
        end
        From this set of spoken labels, we take the most common one and add
          it to node's memory.
    end
end
```

To cluster multiple graphs, we make several minor changes to this algorithm. We choose labels from those spoken by neighbours across every graph instead of just one. Rather

than counting every neighbour's choice of label equally, they are weighted by the weight of the edge between the speaker and listener. When the expression measurements for a pair of genes are negatively correlated, they can have a negative weight in the co-expression graph. We consider edges with a negative weight to be a sign that these vertices should not be clustered together. This is because we assume they are not part of the same functional module, as negative correlation implies that they are not active at the same time. Labels spoken along negative edges are therefore allowed to reduce the cumulative weight of these labels.
To avoid proposing an excessive number of clusters, a node belongs to a cluster only if that cluster's label takes more than a particular fraction of the node's memory, which we call the membership threshold. Once we remove the labels that do not meet this cutoff, every remaining label represents a cluster the node belongs to. The membership weight for the node in each of these clusters is the fraction of the node's remaining memory occupied by the cluster's label.

## 3.4 Including correlation

Our method is intended to include both pairwise interactions and expression data. To see whether our current method does this, we compare the effectiveness of clustering both graphs to that of either graph individually. We simulate ten sets of 500 genes using the parameters from Subsection 3.2.1. Comparing the F1 scores, we find that clustering both the PPI graph and co-expression graph is drastically less accurate than the PPI graph alone (Figure 3.3). Given the poor accuracy of clustering the co-expression graph alone, this is unsurprising. In fact, the clustering using both graphs is almost identical to using co-expression alone (Figure 3.3a).

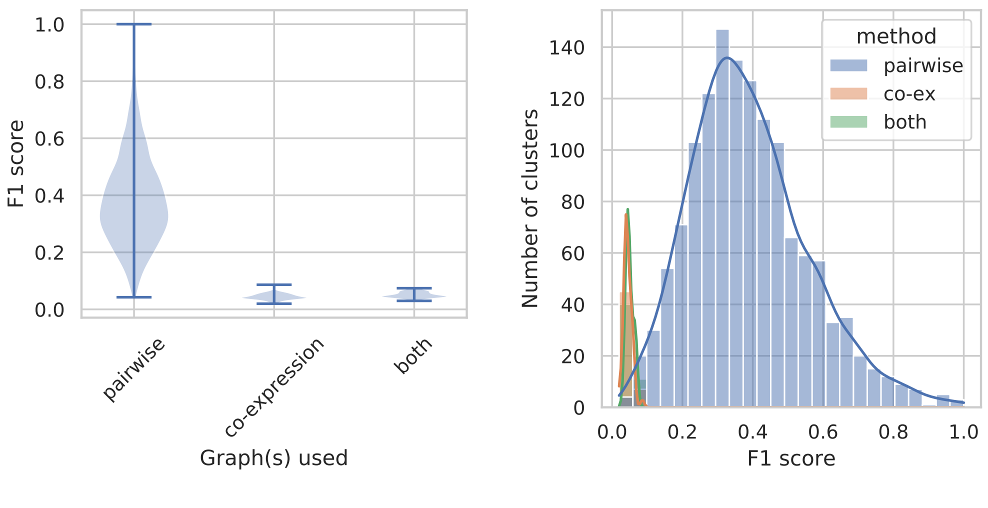
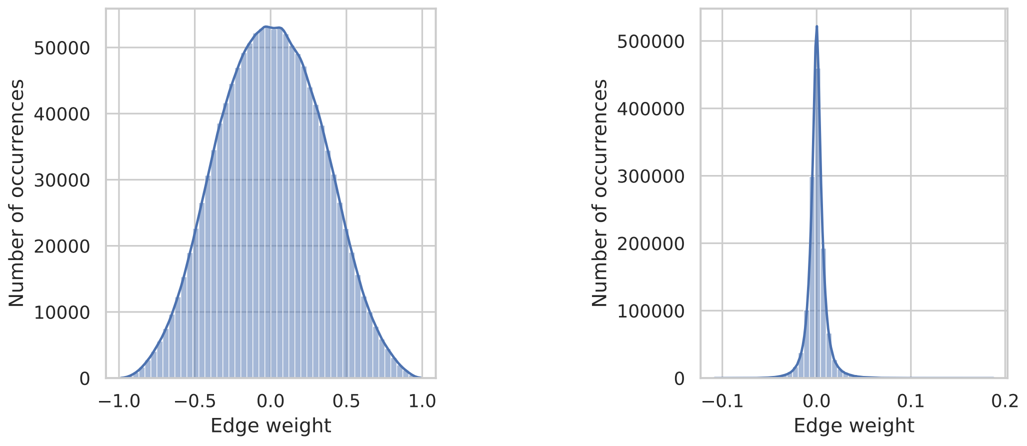


Figure 3.3: Clustering the pairwise interaction graph only, co-expression graph only, and both pairwise and co-expression graphs: (a) F1 score distribution. (b) Number of clusters returned depending on F1 score.

Compared to the PPI graph (Figure 3.1a), the co-expression graph is a complete graph, and therefore includes significantly more edges along which SLPA labels can travel. It is also possible that many of these correlations are spurious, and do not accurately reflect the functional modules present in the simulation. With that in mind, we conclude that the co-expression graph is a poorer source of information than the PPI graph. Every edge in the graph gives an opportunity for a label to travel at each iteration of SLPA. Given its larger number of edges, the co-expression graph also has more influence on the outcome of SLPA than the PPI graph. We attempt to solve this first by adjusting the weights in the co-expression graph (Subsection 3.4.1). Afterwards, we filter out many of the edges in the co-expression graph (Subsection 3.4.2). Finally, accuracy is improved by filtering out clusters that appear to be poor fits after running SLPA (Subsection 3.4.3).

### 3.4.1 Weighted co-expression

We assume that expression of genes that are part of the same functional module will significantly change when the activity of the module changes. With this in mind, highly correlated small changes are likely to be spurious. To avoid giving a significant amount of weight to such changes in our graph, we adjust the co-expression weight according to the standard deviation of the expression of genes involved. The adjusted weight between genes $i$ and $j$ is $w_{adjusted}(i,j) = correlation_{pearson}(i,j) \cdot min(\sigma^2(i), \sigma^2(j))$. The distribution of adjusted and unadjusted co-expression weights is shown in Figure 3.4.



(a) (b)

Figure 3.4: Distribution of (a) unadjusted, and (b) adjusted co-expression edge weights.

After adjusting the co-expression edge weights, we do not see a significant improvement (Figure 3.5). This suggests that the problem is not highly weighted spurious correlations, and we focus instead on reducing the density of the co-expression graph.

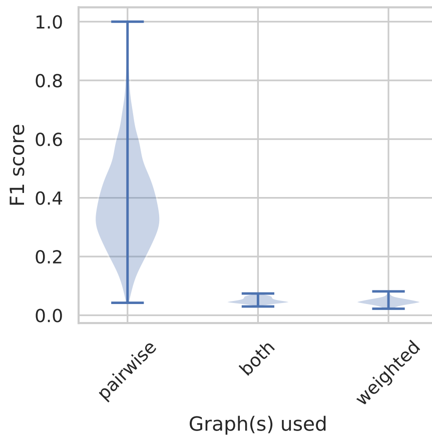


Figure 3.5: F1 score after clustering the pairwise graph alone, both the pairwise and co-expression graphs, and both graphs with adjusted co-expression weights.

### 3.4.2 Filtered Multigraph Method

Including every correlation edge significantly harms performance, perhaps because SLPA is not effective when clustering such a dense graph, even when the weights are mostly small. To make use of the co-expression graph, responsible for the majority of edges, we attempt to remove all but the most significant edges.

Rather than clustering the PPI network directly (Figure 3.6a) or including the complete graph, we include only the significant edges and cluster the resulting graph (Figure 3.6b). We consider an edge between genes $i$ and $j$ to be significant when $min(variance(i), variance(j)) \cdot correlation(i, j)$ is greater than some predefined cutoff. If adding many low weight edges between clusters is the cause of the significant drop in performance in Section 3.4, including only those most likely to be within a cluster should eliminate this problem. By only including an edge between $i$ and $j$ when $min(variance(i), variance(j))$ is high, we only allow edges that represent correlated significant changes. These will be genes whose expression has not just changed in the same direction, but by a significant amount. Since a module becoming active or inactive is expected to result in a significant change in expression, we expect genes in the same module to have such significant correlated changes whenever the activity of the module changes.

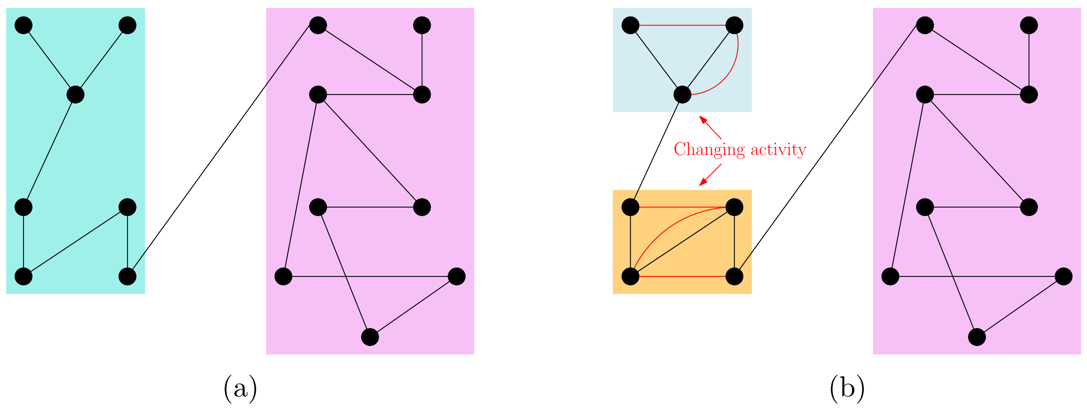


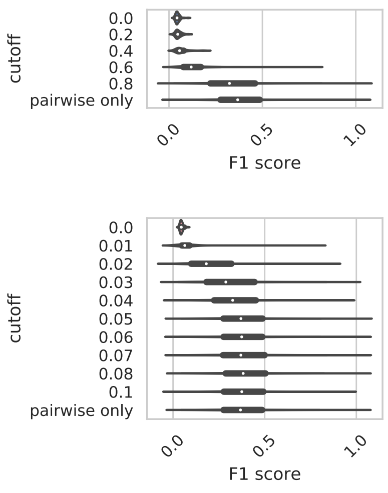
Figure 3.6: Intended effect of including only significant co-expression edges. Black edges represent pairwise interactions, red edges significant co-expression edges.

### Choosing cutoff

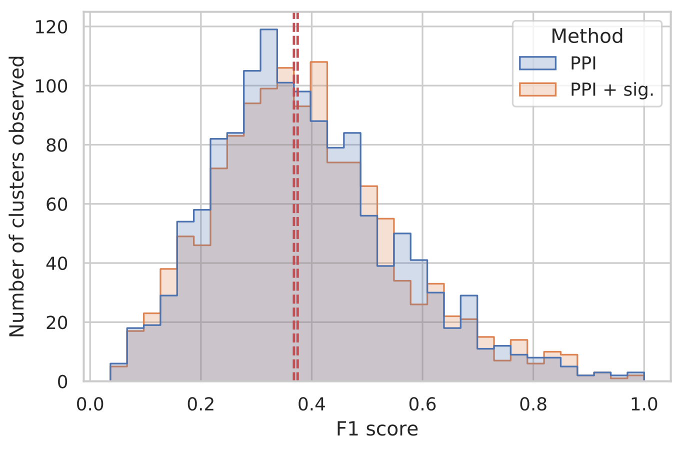
We evaluate the effectiveness of our clustering algorithm as a method for detecting functional modules using the number of correct pairings present in each found cluster. Each cluster is scored in terms of:

- *precision*: Fraction of common cluster members that are correct out of the total number found, summed over all genes in the cluster.
- *recall*: Fraction of common cluster members that are correct out of number expected, adjusted for membership scores.
- *F1 score*: $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$.

We simulated ten sets of 500 genes, using the parameters from Subsection 3.2.1, for cutoff values ranging between zero and one. Given these parameters, each simulated data set has $\approx 70$ partially overlapping clusters with these parameters. We have run this simulation and benchmark with 15,000 genes on an Intel Core i7-6600U CPU in $\approx 1$ minute. The results are shown in Figure 3.7.



(a)



(b)

Figure 3.7: Effect of correlation weight cutoff. (a) F1 score for gradually increasing cutoff, using unadjusted weights above and adjusted weights below. (b) Distribution of F1 scores for the best case (weighted cutoff of 0.06) with (PPI + sig.) and without (PPI) the adjusted-weight co-expression graph.

We find that including only significant edges only marginally improves F1 scores compared to pairwise-only. While the majority of the distribution is the same, there are more clusters in the tails of the distribution (although this is difficult to see in Figure 3.7b). Looking at only clusters with $F1 \geq 0.5$, we find 37 more when the co-expression edges are added, and 23 more when $F1 \geq 0.6$. To remove the poor scoring clusters while preserving the high scoring ones, we now consider filtering the produced clusters by their modularity score.

### 3.4.3 Modularity Method

SLPA is able to propose many overlapping clusters, some of which may be subgraphs of a better cluster. We attempt to filter out only the best-fitting clusters using modularity density, a variation of modularity.

Modularity [63] is a measure of the extent to which a cluster is more connected internally than externally, commonly used as an indication of the quality of a clustering [11, 9, 51]. Modularity suffers from the resolution limit, however [24], in which the size of modules with high scores is limited by the overall size of the graph. To overcome this, and allow scoring of overlapping clusters, we use a modified version, modularity density [12].

We use the following terms to define modularity and modularity density. Given a weighted adjacency matrix $A \in \mathbb{R}^{n \times n}$, clusters $C = \{c_1, c_2, \ldots, c_{|C|}\}$ and membership weights $a_{i,c_k}$, measuring the extent to which node $i$ belongs to cluster $c_k$, we define the following.

$$
\begin{aligned}
|E_c^{in}| &= \frac{1}{2}\sum_{i,j\in c}(a_{i,c}\cdot a_{j,c})A_{ij} && \text{Total weight inside cluster}\\
|E_c^{out}| &= \sum_{i\in c}\sum_{\substack{c'\in C\\ c'\neq c\\ j\in c'}}(a_{i,c}\cdot a_{j,c})A_{ij} && \text{Weight leaving cluster}\\
|E_{c,c'}| &= \sum_{i\in c, j\in c'}(a_{i,c}\cdot a_{j,c'})A_{ij} && \text{Weight between clusters } c \text{ and } c'\\
|E| &= \frac{1}{2}\sum_{i,j\in V}A_{ij} && \text{Total weight in graph}\\
d_c &= \frac{2|E_c^{in}|}{|c|(|c|-1)} && \text{cluster density - prevents favouring large clusters}\\
d_{c,c'} &= \frac{2|E_{c,c'}|}{|c||c'|} && \text{pairwise density between clusters}
\end{aligned}
$$

We then define modularity and modularity density as the following:

$$
\text{(modularity) } Q(C) = \sum_{c\in C}\left[\frac{|E_c^{in}|}{|E|} - \left(\frac{2|E_c^{in}|+|E_c^{out}|}{2|E|}\right)^2\right]
$$

$$
\text{(modularity density) } Q_{ds}^{ov}(C) = \sum_{c\in C}\left[\frac{|E_c^{in}|}{|E|}d_c - \left(\frac{2|E_c^{in}|+|E_c^{out}|}{2|E|}d_c\right)^2 - \sum_{\substack{c'\in C\\ c'\neq c}}\frac{|E_{c,c'}|}{2|E|}d_{c,c'}\right]
$$

Modularity density consists of the following three components.

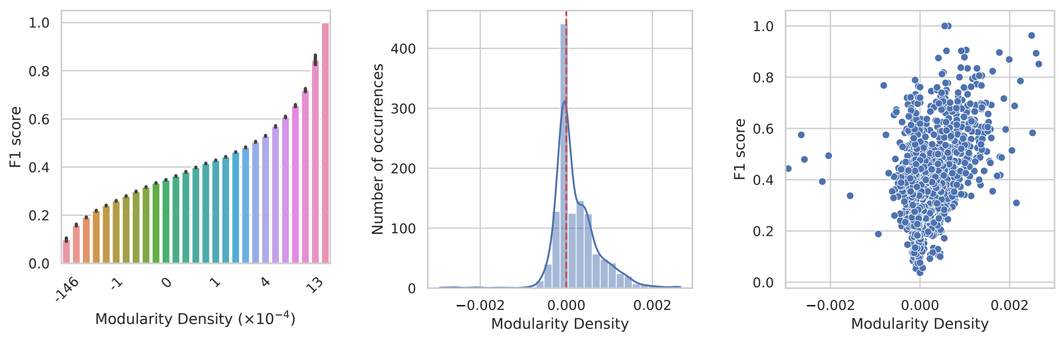
$$
\begin{aligned}
&\frac{|E_c^{in}|}{|E|}d_c && \text{scaled weight inside clusters}\\
&\left(\frac{2|E_c^{in}|+|E_c^{out}|}{2|E|}d_c\right)^2 && \text{Expected weight leaving clusters}\\
&\sum_{\substack{c'\in C\\ c'\neq c}}\frac{|E_{c,c'}|}{2|E|}d_{c,c'} && \text{`Split Penalty'}
\end{aligned}
$$

In terms of these components, the modularity density score represents the degree to which weight inside the clusters exceed expectations, with a penalty for connections between clusters. When using multiple graphs, a cluster's score is the sum of its scores in each graph. We refer to this simply as modularity for the remainder of this chapter.

### Modularity Accuracy

To determine whether modularity is a reasonable filter for SLPA-produced clusters, we run repeated simulations and compare the accuracy of the found clusters with their modularity scores.

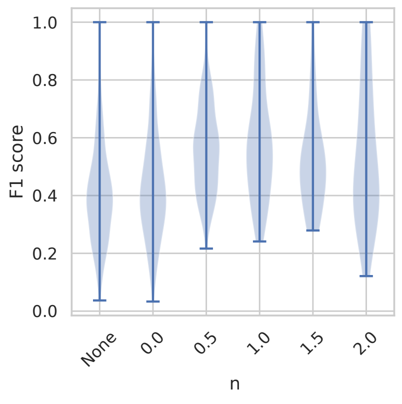
**Score Distribution** We run simulations of 500 genes with an SLPA membership threshold of 0.2. We include only co-expression edges with an adjusted weight greater than or equal to 0.06, adding the unadjusted edge weight to the graph. Repeating this simulation ten times, we calculate the mean F1 score of clusters. The relation between F1 score and modularity, as well as the number of clusters with each modularity score, is shown in Figure 3.8. In our simulations we find that the average F1 score of clusters with high modularity is extremely good (Figure 3.8a) The scores are somewhat scattered, however (Figure 3.8c), and as a result there is no cutoff that guarantees a good F1 score. We can largely rule out clusters with modularity density scores below 0.5 while still retaining a significant number of clusters.

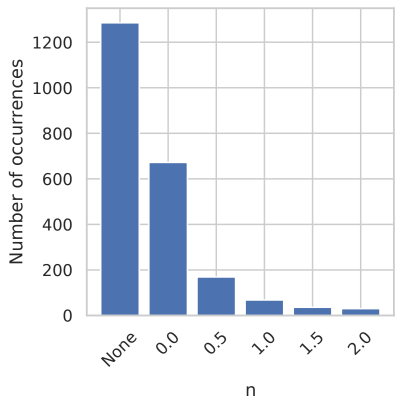


(a) (b) (c)

Figure 3.8: Modularity scores of proposed clusters. (a) Median F1 score. (b) Distribution of modularity scores (red line emphasises $x = 0$). (c) Modularity vs F1 score.

While the average F1 score is very good for clusters with positive modularity, the scores are nonetheless very scattered (Figure 3.8c). We often find clusters with an F1 score of 1.0 are assigned a negative modularity score. Despite the possibility of correct clusters being removed, filtering by modularity is largely beneficial, and we implement it as an option in our package. While a modularity cutoff can be specified manually, a reasonable automatic value would be ideal. Simply removing all clusters with negative scores seems natural, but we find that this does not drastically improve the quality of the clustering (Figure 3.9).

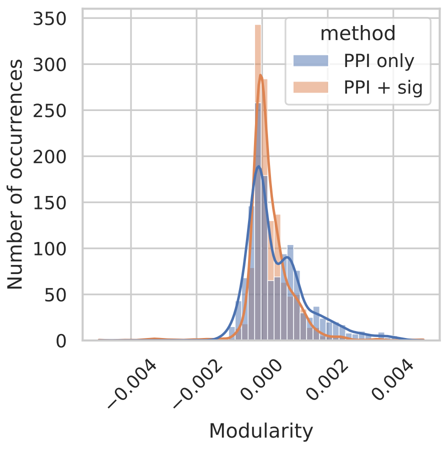


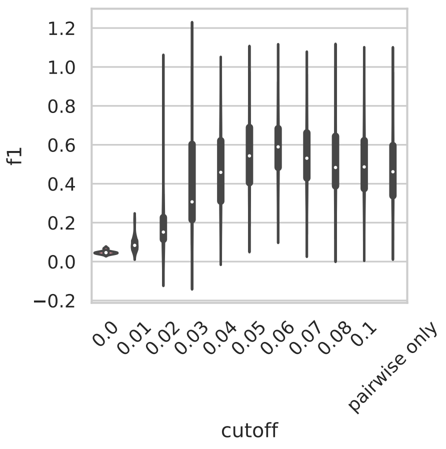


(a) (b)

Figure 3.9: After excluding all clusters with a modularity score below the median + $n$ standard deviations: (a) F1 score. (b) Number of clusters.

We instead include only the clusters with a score greater than median $+ k\sigma$, where $\sigma$ is the standard deviation of the scores of proposed clusters. We find that using $k = 0.5$ provides a good balance in our simulations, although some tuning may be required for other data sets. In ten repeated simulations of 500 genes we find 19 clusters on average, out of 73. While this is a relatively small number of clusters, they are largely correct, with a median F1 score of 0.55.



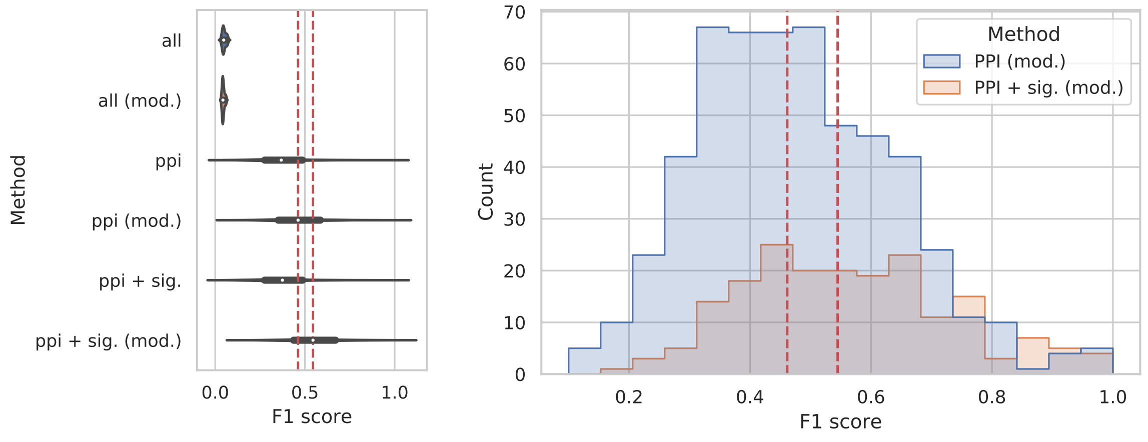


(a) (b)

Figure 3.10: Effect of introducing highly correlated co-expression edges. (a) Distribution of modularity scores. (b) Varying adjusted-weight co-expression cutoff.

Since adding co-expression edges increases the number of high scoring clusters found by SLPA, it is natural to wonder whether there is an effect on the distribution of modularity scores. Repeating the test from Section 3.4.2 we find that modularity is generally reduced when co-expression is added (Figure 3.10a). After filtering out all clusters with modularity less than $median + \frac{\sigma}{2}$ we are left with a better selection than if we only use the pairwise interaction graph (Figure 3.10b). It is worth noting that the additional edges affect not only the clusters found by SLPA, but the modularity scores clusters are given. Based on these results, we use co-expression edges with an adjusted weight cutoff of 0.06 for our simulated data in the remainder of this section.

**Modularity filtering combines well with co-ex filtering** Modularity filtering reduces the number of poor scoring clusters we find with and without co-expression edges (Figure 3.11). While adding co-expression edges is not a clear improvement over pairwise only, combining co-expression edges with modularity filtering we are able to remove many inaccurate clusters, significantly improving performance.



(a) (b)

Figure 3.11: Significant-only co-expression edges vs. modularity filter. Median values for two highest scoring sets are shown as dashed lines.

## 3.5 Active Module Detection

The clustering method we have developed is intended to accurately summarise differences in gene activity across tissues, by presenting that activity in terms of clusters. To do so, we return a list of clusters that are active for each tissue present in the expression data. This is shown for one of our simulated data sets in Figure 3.12. The reconstructed set of active clusters is reasonably accurate, there are only seven mis-classified genes out of 100 across both graphs.

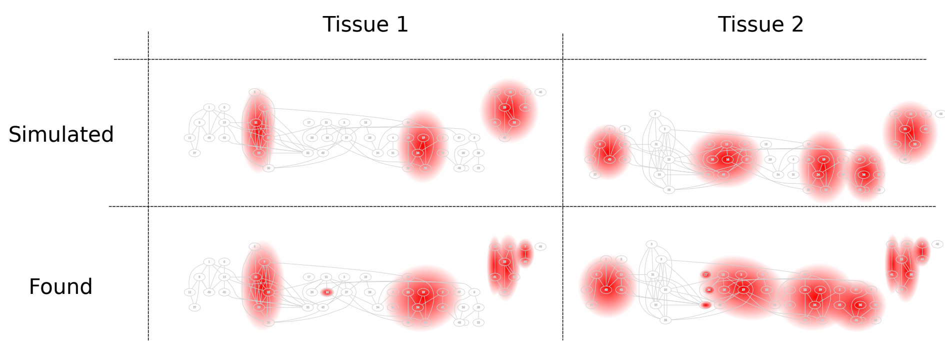


Figure 3.12: Ground truth vs. found cluster activity in two different tissues. Combined pairwise interaction and significant co-expression graph is present in grey. Active clusters are highlighted in red in both cases. Inactive clusters are not shown.

For a given tissue we say a found cluster is active if the mean activity of its nodes is greater than 0.5. We say a node is truly active if the sum of the activity of the true simulated clusters it is in, adjusted for its membership in each cluster, is greater than 0.5:

$$\sum_{c:v\in c} activity_c \cdot a_{v,c} > 0.5$$

, where $activity_c$ is zero if cluster $c$ is inactive, and one if cluster $c$ is active.
Again simulating ten sets of 500 genes, we find that the vast majority of clusters are classified as either active or inactive correctly. We looked at the fraction of genes within each cluster that were correctly classified, and show the distribution in Figure 3.13. Out of 17, 750 clusters we find that 71.9% have correctly classified over 90% of their genes.

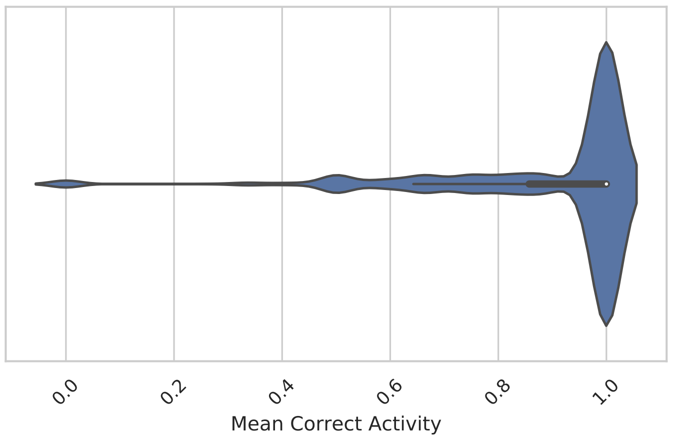


Figure 3.13: Activity accuracy of each cluster

## 3.6 Discussion

We present a method for detecting functional modules, using an arbitrary pairwise interaction graph, and normalised co-expression data. To effectively use co-expression edges with SLPA, we first need to filter out the majority of low weight edges. We achieve the best performance when we remove any that have a low adjusted weight, which is the product of the Pearson correlation and minimum variance of the two connected genes. Once this is done, SLPA appears to work in our simulations as a method of functional module detection, and has an exceptionally quick run time.
Performance is further improved when we filter out many of the less accurate clusters proposed by SLPA, by removing those with a low modularity density score. We specifically propose removing those with a score less than the median plus half of one standard deviation, a cutoff that performs well in our simulations. Once these measures are in place, we are able to identify underlying clusters present in both the pairwise network and expression data. Classifying clusters as either active or inactive, the particular focus of this method, is achieved with particularly high accuracy. The vast majority of genes are placed into a cluster with the correct activity.

### 3.6.1 Limitations and Future Work

We used two methods to improve the quality of the clustering over the one SLPA produces on the pairwise interaction graph. Both of these, including co-expression edges above a certain $weight \cdot variance$, and filtering out clusters with a low modularity score, require some tuning. The default modularity cutoff is chosen according to the standard deviation of modularity scores present, and may be reasonable for other data sets. This value does not reliably produce clusters that cover the entire graph, even when they are present, and may still need manual tuning. It might be possible to solve this by automatically reducing the cutoff until a certain fraction of nodes, 80% for example, are included. This way we could guarantee that the clustering summarises the majority of the data, while still using only the best clusters available. Similarly, the co-expression cutoff requires manual tuning. In this case, we have used a cutoff that is specific to our data, and a more general method is required to use this effectively in other cases. It may be possible to solve this by choosing a value that is again based on the standard deviation of adjusted co-expression weights, or one that maximises modularity with cross-validation.
We frequently find clusters with our method that are correct components of a larger cluster. It may be possible to detect and merge these automatically, by merging adjacent clusters whenever this improves the total modularity score of the graph. All of our simulations were performed with the same parameters, and many of the properties known to be present in biological networks were not present in our simulations. It would be informative to generate a wide variety of graphs, particularly including those with a similar structure to real biological networks, and compare performance as parameters are changed. The general method can also be extended for an arbitrary number of pairwise interaction graphs, as long as they represent the same clusters. For example, it may be beneficial to include both protein interactions, and epistatic pairs.

We have not attempted to implement any of these proposals, as they go beyond the scope of this thesis. They may be worthwhile subjects for future research.

# Summary

Exploiting the large number of off-target effects typically present in siRNA screens, we presented a model for large-scale prediction of epistasis between pairs of genes in Chapter 1. This model was evaluated using two different statistical methods for selecting non-zero effects, `xyz` and `glinternet`. We found that, while `glinternet` is able to accurately infer interactions, its run time does not scale well to siRNA screens of more than $1,000$ genes, including off-targets. Our alternative method `xyz` is able to run quickly on much larger data, but suffers from a loss of accuracy beyond $1,000$ genes.
We introduced a new lasso-based method in Chapter 2 that has accuracy comparable to `glinternet`, and is able to process exome-scale data. Beginning with an efficient serial algorithm for lasso regression, we investigated a variety of methods for parallelisation. This was based on the idea that parameters for non-overlapping columns can be updated in parallel. We investigated guaranteeing this restriction by finding sets of non-overlapping columns, limiting simultaneous updates to parameters of mostly non-overlapping columns, and relying on the sparsity of the matrix by randomly shuffling columns. While all three methods improved performance over the serial algorithm, random shuffling requires no preprocessing of the interaction matrix, and has the best performance overall. Substantial performance improvements were obtained by using a sparse offset based representation of the input pairwise interaction matrix, in which columns were delta encoded and compressed with Simple-8b. This allows exome-scale data analysis on a sufficiently large shared memory system, without requiring a GPU, in a matter of hours. This can be further improved by restricting the allowed distance between interacting genes along the gene sequence. If interactions are restricted to those within 100 positions of each other, we could search $n = 200,000$ siRNAs and $p = 30,000$ genes using only 4GB of memory. Comparing our shuffled implementation to `glinternet`, we were able to achieve marginally better prediction of genetic interactions in several orders of magnitude less time.
Finally, in Chapter 3 we considered a more general model of interactions, interaction networks, and developed a method for inferring functional modules from a combination of pairwise interaction and gene expression data. Our method formed a correlated expression graph containing edges between only highly correlated and significantly varying genes. This graph and the graph of pairwise interactions were simultaneously clustered using the SLPA algorithm, removing all clusters with a low modularity density score afterwards. The resulting method was able to effectively find functional modules in our simulated data, and was particularly accurate in placing active or inactive genes into active or inactive clusters.

## Acknowledgements

I would like to thank the following people for their contributions, without which this work would not have been possible. My supervisors, Alex Gavryushkin, Matthieu Vignes, and Zhiyi Huang, for their ongoing advice throughout this project. The co-authors of Chapter 1, Fabian Schmich, Ewa Szczurek, Jeremy Jenkins, Niko Beerenwinkel, and Alex Gavryushkin, for their contributions to this paper. Anaïs Baudot and Élisabeth Remy for their helpful discussions of biological networks and clustering, and Paul Gardner for discussions of genetic interaction distance. Finally, Lena Collienne for invaluable proof-reading of this thesis.

# Bibliography

[1] Vo Ngoc Anh and Alistair Moffat. “Index Compression Using 64-Bit Words”. In: *Software: Practice and Experience* 40.2 (2010), pp. 131–147. ISSN: 1097-024X. DOI: 10.1002/spe.948. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.948 (visited on 08/08/2019).

[2] Vo Ngoc Anh and Alistair Moffat. “Inverted Index Compression Using Word-Aligned Binary Codes”. In: *Information Retrieval* 8.1 (Jan. 1, 2005), pp. 151–166. ISSN: 1573-7659. DOI: 10.1023/B:INRT.0000048490.99518.5c. URL: https://doi.org/10.1023/B:INRT.0000048490.99518.5c (visited on 08/30/2020).

[3] Alan Ashworth, Christopher J Lord, and Jorge S Reis-Filho. “Genetic Interactions in Cancer Progression and Treatment”. In: *Cell* (2011).

[4] Albert-László Barabási and Zoltán N. Oltvai. “Network Biology: Understanding the Cell’s Functional Organization”. en. In: *Nature Reviews Genetics* 5.2 (Feb. 2004), pp. 101–113. ISSN: 1471-0064. DOI: 10.1038/nrg1272.

[5] N Beerenwinkel, L Pachter, and B Sturmfels. “Epistasis and Shapes of Fitness Landscapes”. In: *Statistica Sinica* (2007).

[6] Yoav Benjamini and Yosef Hochberg. “Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing”. en. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 57.1 (1995), pp. 289–300. ISSN: 2517-6161. DOI: 10.1111/j.2517-6161.1995.tb02031.x.

[7] Laura Bennett et al. “Detection of Composite Communities in Multiplex Biological Networks”. en. In: *Scientific Reports* 5 (May 2015), p. 10345. ISSN: 2045-2322. DOI: 10.1038/srep10345.

[8] Joseph K. Bradley et al. “Parallel Coordinate Descent for L1-Regularized Loss Minimization”. In: (May 26, 2011). arXiv: 1105.5379 [cs, math]. URL: http://arxiv.org/abs/1105.5379 (visited on 07/14/2019).

[9] Jie Cao et al. “Weighted Modularity Optimization for Crisp and Fuzzy Community Detection in Large-Scale Networks”. In: *Physica A: Statistical Mechanics and its Applications* 462 (Nov. 2016), pp. 386–395. ISSN: 0378-4371. DOI: 10.1016/j.physa.2016.06.113.

[10] Denise A Chan and Amato J Giaccia. "Harnessing Synthetic Lethal Interactions in Anticancer Drug Discovery." In: *Nature Reviews Drug Discovery* (2011).

[11] M. Chen, K. Kuzmin, and B. K. Szymanski. "Community Detection via Maximization of Modularity and Its Variants". In: *IEEE Transactions on Computational Social Systems* 1.1 (Mar. 2014), pp. 46–65. ISSN: 2329-924X. DOI: 10.1109/TCSS.2014.2307458.

[12] M. Chen, K. Kuzmin, and B. K. Szymanski. "Extension of Modularity Density for Overlapping Community Structure". In: *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*. Aug. 2014, pp. 856–863. DOI: 10.1109/ASONAM.2014.6921686.

[13] Michael Chichignoud, Johannes Lederer, and Martin J Wainwright. "A Practical Scheme and Fast Algorithm to Tune the Lasso With Optimality Guarantees". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 8162–8181.

[14] M. Clamp et al. "Distinguishing Protein-Coding and Noncoding Genes in the Human Genome". In: *Proceedings of the National Academy of Sciences* 104.49 (Dec. 4, 2007), pp. 19428–19433. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.0709013104. URL: http://www.pnas.org/cgi/doi/10.1073/pnas.0709013104 (visited on 08/31/2020).

[15] H. J. Cordell. "Epistasis: What It Means, What It Doesn't Mean, and Statistical Methods to Detect It in Humans". In: *Human Molecular Genetics* 11.20 (Oct. 1, 2002), pp. 2463–2468. ISSN: 14602083. DOI: 10.1093/hmg/11.20.2463. URL: https://academic.oup.com/hmg/article-lookup/doi/10.1093/hmg/11.20.2463 (visited on 07/14/2019).

[16] Heather J Cordell. "Epistasis: What It Means, What It Doesn't Mean, and Statistical Methods to Detect It in Humans." In: *Human molecular genetics* (2002).

[17] Michael Costanzo et al. "The Genetic Landscape of a Cell." In: *Science* (2010).

[18] J. Arjan G. M. de Visser, Tim F. Cooper, and Santiago F. Elena. "The Causes of Epistasis". In: *Proceedings of the Royal Society B: Biological Sciences* 278.1725 (Dec. 2011), pp. 3617–3624. DOI: 10.1098/rspb.2011.1537.

[19] Broad DepMap. *DepMap 20Q2 Public*. June 2020. DOI: 10.6084/m9.figshare.12280541.v4.

[20] Gilles Didier, Christine Brun, and Anaïs Baudot. "Identifying Communities from Multiplex Biological Networks". In: *PeerJ* 3 (Dec. 2015). ISSN: 2167-8359. DOI: 10.7717/peerj.1525.

[21] Kieran Elmes et al. "Learning Epistatic Gene Interactions from Perturbation Screens". In: *bioRxiv* (Aug. 25, 2020), p. 2020.08.24.264713. DOI: 10.1101/2020.08.24.264713. URL: https://www.biorxiv.org/content/10.1101/2020.08.24.264713v1 (visited on 08/31/2020).

[22] Thomas Force and Kyle L Kolaja. "Cardiotoxicity of Kinase Inhibitors: The Prediction and Translation of Preclinical Models to Clinical Outcomes." In: *Nature Reviews Drug Discovery* (2011).

[23] Jochen Förster et al. "Genome-Scale Reconstruction of the Saccharomyces Cerevisiae Metabolic Network". en. In: *Genome Research* 13.2 (Jan. 2003), pp. 244–253. ISSN: 1088-9051, 1549-5469. DOI: 10.1101/gr.234503.

[24] Santo Fortunato and Marc Barthélemy. "Resolution Limit in Community Detection". en. In: *Proceedings of the National Academy of Sciences* 104.1 (Jan. 2007), pp. 36–41. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.0605965104.

[25] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. "Regularization Paths for Generalized Linear Models via Coordinate Descent". In: *Journal of statistical software* 33.1 (2010), pp. 1–22. ISSN: 1548-7660. pmid: 20808728. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2929880/ (visited on 05/27/2020).

[26] Wenjiang J. Fu. "Penalized Regressions: The Bridge versus the Lasso". In: *Journal of Computational and Graphical Statistics* 7.3 (Sept. 1998), pp. 397–416. ISSN: 1061-8600, 1537-2715. DOI: 10.1080/10618600.1998.10474784. URL: http://www.tandfonline.com/doi/abs/10.1080/10618600.1998.10474784 (visited on 06/19/2020).

[27] David M Garcia et al. "Weak Seed-Pairing Stability and High Target-Site Abundance Decrease the Proficiency of Lsy-6 and Other microRNAs". In: *Nature Structural & Molecular Biology* (2011).

[28] Anne-Claude Gavin et al. "Functional Organization of the Yeast Proteome by Systematic Analysis of Protein Complexes". en. In: *Nature* 415.6868 (Jan. 2002), pp. 141–147. ISSN: 1476-4687. DOI: 10.1038/415141a.

[29] Douglas Hanahan and Robert A Weinberg. "Hallmarks of Cancer: The next Generation." In: *Cell* (2011).

[30] Steve Harenberg et al. "Community Detection in Large-Scale Networks: A Survey and Empirical Evaluation". en. In: *WIREs Computational Statistics* 6.6 (2014), pp. 426–439. ISSN: 1939-0068. DOI: 10.1002/wics.1319.

[31] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer New York, 2009.

[32] Caitriona Holohan et al. "Cancer Drug Resistance: An Evolving Paradigm." In: *Nature Reviews. Cancer* (2013).

[33] J. Huggett et al. "Real-Time RT-PCR Normalisation; Strategies and Considerations". en. In: *Genes & Immunity* 6.4 (June 2005), pp. 279–284. ISSN: 1476-5470. DOI: 10.1038/sj.gene.6364190.

[34] T. Ito et al. “A Comprehensive Two-Hybrid Analysis to Explore the Yeast Protein Interactome”. en. In: *Proceedings of the National Academy of Sciences* 98.8 (Apr. 2001), pp. 4569–4574. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.061034498.

[35] Aimee L. Jackson et al. “Widespread siRNA ”off-Target” Transcript Silencing Mediated by Seed Region Sequence Complementarity”. In: *RNA* 12.7 (July 2006), pp. 1179–1187. ISSN: 1355-8382, 1469-9001. DOI: 10.1261/rna.25706.

[36] Laurent Jacob, Guillaume Obozinski, and Jean-Philippe Vert. “Group Lasso with Overlap and Graph Lasso”. In: *ICML ’09*. 2009.

[37] H. Jeong et al. “Lethality and Centrality in Protein Networks”. en. In: *Nature* 411.6833 (May 2001), pp. 41–42. ISSN: 1476-4687. DOI: 10.1038/35075138.

[38] Livnat Jerby-Arnon et al. “Predicting Cancer-Specific Vulnerability via Data-Driven Detection of Synthetic Lethality”. English. In: *Cell* 158.5 (Aug. 2014), pp. 1199–1209.

[39] J. Ji et al. “Survey: Functional Module Detection from Protein-Protein Interaction Networks”. In: *IEEE Transactions on Knowledge and Data Engineering* 26.2 (Feb. 2014), pp. 261–277. ISSN: 1041-4347. DOI: 10.1109/TKDE.2012.225.

[40] Junzhong Ji et al. “Survey: Functional Module Detection from Protein-Protein Interaction Networks”. In: *IEEE Transactions on Knowledge and Data Engineering* 26.2 (Feb. 2014), pp. 261–277. ISSN: 1558-2191. DOI: 10.1109/TKDE.2012.225.

[41] Daxin Jiang, Chun Tang, and Aidong Zhang. “Cluster Analysis for Gene Expression Data: A Survey”. In: *IEEE Transactions on Knowledge and Data Engineering* 16.11 (Nov. 2004), pp. 1370–1386. ISSN: 1558-2191. DOI: 10.1109/TKDE.2004.68.

[42] William G Kaelin. “The Concept of Synthetic Lethality in the Context of Anticancer Therapy.” In: *Nature Reviews. Cancer* (2005).

[43] Atanas Kamburov et al. “ConsensusPathDB-a Database for Integrating Human Functional Interaction Networks”. In: *Nucleic Acids Research* 37 (Jan. 2009), pp. D623–D628. ISSN: 0305-1048. DOI: 10.1093/nar/gkn698.

[44] Kumaran Kandasamy et al. “NetPath: A Public Resource of Curated Signal Transduction Pathways”. In: *Genome Biology* 11.1 (Jan. 2010), R3. ISSN: 1474-760X. DOI: 10.1186/gb-2010-11-1-r3.

[45] Seketoulie Keretsu and Rosy Sarmah. “Weighted Edge Based Clustering to Identify Protein Complexes in Protein–Protein Interaction Networks Incorporating Gene Expression Profile”. en. In: *Computational Biology and Chemistry* 65 (Dec. 2016), pp. 69–79. ISSN: 1476-9271. DOI: 10.1016/j.compbiolchem.2016.10.001.

[46] K. Kuzmin, S. Y. Shah, and B. K. Szymanski. “Parallel Overlapping Community Detection with SLPA”. In: *2013 International Conference on Social Computing*. Sept. 2013, pp. 204–212. DOI: 10.1109/SocialCom.2013.37.

[47] Daniel Langr et al. “Algorithm 947: Paraperm—Parallel Generation of Random Permutations with MPI”. In: *ACM Transactions on Mathematical Software* 41.1 (Oct. 27, 2014), 5:1–5:26. ISSN: 0098-3500. DOI: 10.1145/2669372. URL: https://doi.org/10.1145/2669372 (visited on 08/19/2020).

[48] Christina Laufer et al. “Mapping Genetic Interactions in Human Cancer Cells with RNAi and Multiparametric Phenotyping.” In: *Nat Meth* (2013).

[49] D. Lemire and L. Boytsov. “Decoding Billions of Integers per Second through Vectorization”. In: *Software: Practice and Experience* 45.1 (2015), pp. 1–29. ISSN: 1097-024X. DOI: 10.1002/spe.2203. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2203 (visited on 07/13/2020).

[50] Benjamin P. Lewis et al. “Prediction of Mammalian MicroRNA Targets”. In: *Cell* 115.7 (Dec. 2003), pp. 787–798. ISSN: 0092-8674. DOI: 10.1016/S0092-8674(03)01018-3.

[51] Zhenping Li et al. “Quantitative Function for Community Detection”. In: *Physical Review E* 77.3 (Mar. 2008), p. 036109. DOI: 10.1103/PhysRevE.77.036109.

[52] Lifan Liang et al. “Integrating Data and Knowledge to Identify Functional Modules of Genes: A Multilayer Approach”. In: *BMC Bioinformatics* 20 (May 2019). ISSN: 1471-2105. DOI: 10.1186/s12859-019-2800-y.

[53] Michael Lim and Trevor Hastie. “Learning Interactions via Hierarchical Group-Lasso Regularization”. In: *Journal of Computational and Graphical Statistics* 24.3 (July 3, 2015), pp. 627–654. ISSN: 1061-8600. DOI: 10.1080/10618600.2014.938812. pmid: 26759522. URL: https://doi.org/10.1080/10618600.2014.938812 (visited on 08/31/2020).

[54] Hong-Wu Ma, Jan Buer, and An-Ping Zeng. “Hierarchical Structure and Modules in the Escherichia Coli Transcriptional Regulatory Network Revealed by a New Top-down Approach”. en. In: *BMC Bioinformatics* 5.1 (Dec. 2004), p. 199. ISSN: 1471-2105. DOI: 10.1186/1471-2105-5-199.

[55] Hong-Wu Ma et al. “Decomposition of Metabolic Network into Functional Modules Based on the Global Connectivity Structure of Reaction Graph”. en. In: *Bioinformatics* 20.12 (Aug. 2004), pp. 1870–1876. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bth167.

[56] X. Ma, P. G. Sun, and Z. Zhang. “An Integrative Framework for Protein Interaction Network and Methylation Data to Discover Epigenetic Modules”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (2018), pp. 1–1. DOI: 10.1109/TCBB.2018.2831666.

[57] Antonio Mallia, Michał Siedlaczek, and Torsten Suel. "An Experimental Study of Index Compression and DAAT Query Processing Methods". In: *Advances in Information Retrieval.* Ed. by Leif Azzopardi et al. Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 353–368. ISBN: 978-3-030-15712-8.

[58] E. Robert McDonald et al. "Project DRIVE: A Compendium of Cancer Dependencies and Synthetic Lethal Relationships Uncovered by Large-Scale, Deep RNAi Screening". In: *Cell* 170.3 (July 2017), 577–592.e10. ISSN: 00928674. DOI: `10.1016/j.cell.2017.07.005`.

[59] George Montanez and Young-Rae Cho. "Predicting False Positives of Protein-Protein Interaction Data by Semantic Similarity Measures§". In: *Current Bioinformatics* 8.3 (July 2013), pp. 339–346.

[60] F Mosteller and J W Tukey. "Data Analysis and Regression: A Second Course in Statistics." In: *Addison-Wesley Series in Behavioral Science: Quantitative Methods* (1977).

[61] Patrick Y Muller et al. "Short Technical Report Processing of Gene Expression Data Generated by Quantitative Real-Time RT-PCR". In: *Biotechniques* 32.6 (2002), pp. 1372–1379.

[62] Ramachandra Nanjegowda et al. "Scalability Evaluation of Barrier Algorithms for OpenMP". In: *Evolving OpenMP in an Age of Extreme Parallelism.* Ed. by Matthias S. Müller, Bronis R. de Supinski, and Barbara M. Chapman. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 42–52. ISBN: 978-3-642-02303-3. DOI: `10.1007/978-3-642-02303-3_4`.

[63] M. E. J. Newman. "Modularity and Community Structure in Networks". In: *Proceedings of the National Academy of Sciences of the United States of America* 103.23 (June 2006), pp. 8577–8582. ISSN: 0027-8424. DOI: `10.1073/pnas.0601602103`.

[64] Nigel J. O'Neil, Melanie L. Bailey, and Philip Hieter. "Synthetic Lethality and Cancer". In: *Nature Reviews Genetics* 18.1010 (Oct. 2017), pp. 613–623. ISSN: 1471-0064. DOI: `10.1038/nrg.2017.47`.

[65] H Allen Orr. "Fitness and Its Role in Evolutionary Genetics". English. In: *Nature Reviews. Genetics* 10.8 (Aug. 2009), pp. 531–539.

[66] Rose Oughtred et al. "The BioGRID Interaction Database: 2019 Update". en. In: *Nucleic Acids Research* 47.D1 (Jan. 2019), pp. D529–D541. ISSN: 0305-1048. DOI: `10.1093/nar/gky1079`.

[67] Arzucan Özgür et al. "Identifying Gene-Disease Associations Using Centrality on a Literature Mined Gene-Interaction Network". en. In: *Bioinformatics* 24.13 (July 2008), pp. i277–i285. ISSN: 1367-4803. DOI: `10.1093/bioinformatics/btn182`.

[68] Marina S. Paez, Arash A. Amini, and Lizhen Lin. “Hierarchical Stochastic Block Model for Community Detection in Multiplex Networks”. In: *arXiv:1904.05330 [cs, stat]* (Mar. 2019). arXiv: `1904.05330 [cs, stat]`.

[69] Ashwini Patil, Kengo Kinoshita, and Haruki Nakamura. “Hub Promiscuity in Protein-Protein Interaction Networks”. en. In: *International Journal of Molecular Sciences* 11.4 (Apr. 2010), pp. 1930–1943. DOI: `10.3390/ijms11041930`.

[70] powturbo. *Powturbo/TurboPFor-Integer-Compression.* July 9, 2020. URL: `https://github.com/powturbo/TurboPFor-Integer-Compression` (visited on 07/09/2020).

[71] Olga Puchta et al. “Network of Epistatic Interactions within a Yeast snoRNA”. English. In: *Science* 352.6287 (May 2016), pp. 840–844.

[72] Olga Puchta et al. “Network of Epistatic Interactions within a Yeast snoRNA”. In: *Science* 352.6287 (May 13, 2016), pp. 840–844. ISSN: 0036-8075, 1095-9203. DOI: `10.1126/science.aaf0965`. pmid: `27080103`. URL: `https://science.sciencemag.org/content/352/6287/840` (visited on 08/13/2020).

[73] Visakh R. and K. A. Abdul Nazeer. “Multi-Network Approach to Identify Differentially Methylated Gene Communities in Cancer”. In: *Gene* 697 (May 2019), pp. 227–237. ISSN: 0378-1119. DOI: `10.1016/j.gene.2019.02.007`.

[74] Pauli Rämö et al. “Simultaneous Analysis of Large-Scale RNAi Screens for Pathogen Entry”. In: *BMC Genomics* 15.1 (Dec. 2014), p. 1162. ISSN: 1471-2164. DOI: `10.1186/1471-2164-15-1162`.

[75] Rebecca Sanders et al. “Considerations for Accurate Gene Expression Measurement by Reverse Transcription Quantitative PCR When Analysing Clinical Samples”. en. In: *Analytical and Bioanalytical Chemistry* 406.26 (Oct. 2014), pp. 6471–6483. ISSN: 1618-2650. DOI: `10.1007/s00216-014-7857-x`.

[76] Satu Elisa Schaeffer. “Graph Clustering”. In: *Computer Science Review* 1.1 (Aug. 2007), pp. 27–64. ISSN: 1574-0137. DOI: `10.1016/j.cosrev.2007.05.001`.

[77] Benjamin Schlegel, Rainer Gemulla, and Wolfgang Lehner. “Fast Integer Compression Using SIMD Instructions”. In: *Proceedings of the Sixth International Workshop on Data Management on New Hardware - DaMoN ’10.* The Sixth International Workshop. Indianapolis, Indiana: ACM Press, 2010, pp. 34–40. ISBN: 978-1-4503-0189-3. DOI: `10.1145/1869389.1869394`. URL: `http://portal.acm.org/citation.cfm?doid=1869389.1869394` (visited on 07/13/2020).

[78] Fabian Schmich et al. “gespeR: A Statistical Model for Deconvoluting off-Target-Confounded RNA Interference Screens.” In: *Genome Biology* (2015).

[79] Nikolaus Schultz et al. “Off-Target Effects Dominate a Large-Scale RNAi Screen for Modulators of the TGF-$\beta$ Pathway and Reveal microRNA Regulation of TGFBR2.” In: *Silence* (2011).

[80] Saurabh Singh, Ajit S. Narang, and Ram I. Mahato. “Subcellular Fate and Off-Target Effects of siRNA, shRNA, and miRNA”. In: *Pharmaceutical Research* 28.12 (Dec. 2011), pp. 2996–3015. ISSN: 1573-904X. DOI: `10.1007/s11095-011-0608-1`.

[81] Victor Spirin and Leonid A. Mirny. “Protein Complexes and Functional Modules in Molecular Networks”. en. In: *Proceedings of the National Academy of Sciences* 100.21 (Oct. 2003), pp. 12123–12128. ISSN: 0027-8424, 1091-6490. DOI: `10.1073/pnas.2032324100`.

[82] Sumana Srivatsa et al. “Improved Pathway Reconstruction from RNA Interference Screens by Exploiting Off-Target Effects”. In: *Bioinformatics* 34.13 (July 2018), pp. i519–i527. ISSN: 1367-4803. DOI: `10.1093/bioinformatics/bty240`.

[83] Michael Steckel et al. “Determination of Synthetic Lethal Interactions in KRAS Oncogene-Dependent Cancer Cells Reveals Novel Therapeutic Targeting Strategies.” In: *Cell Research* (2012).

[84] Ulrich Stelzl et al. “A Human Protein-Protein Interaction Network: A Resource for Annotating the Proteome”. en. In: *Cell* 122.6 (Sept. 2005), pp. 957–968. ISSN: 0092-8674. DOI: `10.1016/j.cell.2005.08.029`.

[85] *STRING: Functional Protein Association Networks*. URL: `https://string-db.org/cgi/about.pl` (visited on 07/22/2020).

[86] Damian Szklarczyk et al. “The STRING Database in 2017: Quality-Controlled Protein–Protein Association Networks, Made Broadly Accessible”. In: *Nucleic Acids Research* 45.D1 (Jan. 4, 2017), pp. D362–D368. ISSN: 0305-1048. DOI: `10.1093/nar/gkw937`. URL: `https://academic.oup.com/nar/article/45/D1/D362/2290901` (visited on 07/22/2020).

[87] Reiko Tanaka, Tau-Mu Yi, and John Doyle. “Some Protein Interaction Data Do Not Exhibit Power Law Statistics”. en. In: *FEBS Letters* 579.23 (Sept. 2005), pp. 5140–5144. ISSN: 0014-5793. DOI: `10.1016/j.febslet.2005.08.024`.

[88] Wei Tang et al. “A Genome-Wide RNAi Screen for Wnt/Beta-Catenin Pathway Components Identifies Unexpected Roles for TCF Transcription Factors in Cancer.” In: *Proceedings of The National Academy Of Sciences Of The United States Of America* (2008).

[89] Gian-Andrea Thanei, Nicolai Meinshausen, and Rajen D Shah. “The Xyz Algorithm for Fast Interaction Search in High-Dimensional Data”. In: *Journal of Machine Learning Research* 19.1 (Jan. 2018), pp. 1343–1384.

[90] Robert Tibshirani. “Regression Shrinkage and Selection Via the Lasso”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288. ISSN: 2517-6161. DOI: `10.1111/j.2517-6161.1996.tb02080.x`. URL: `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1996.tb02080.x` (visited on 08/29/2020).

[91] Jerzy Tiuryn and Ewa Szczurek. “Learning Signaling Networks from Combinatorial Perturbations by Exploiting siRNA Off-Target Effects”. en. In: *Bioinformatics* 35.14 (July 2019), pp. i605–i614. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btz334.

[92] Andrew Trotman and Jimmy Lin. “In Vacuo and In Situ Evaluation of SIMD Codecs”. In: *Proceedings of the 21st Australasian Document Computing Symposium.* ADCS ’16. Caulfield, VIC, Australia: Association for Computing Machinery, Dec. 5, 2016, pp. 1–8. ISBN: 978-1-4503-4865-2. DOI: 10.1145/3015022.3015023. URL: https://doi.org/10.1145/3015022.3015023 (visited on 07/08/2020).

[93] Sara Van de Geer. “The Deterministic Lasso”. In: 2007.

[94] Heather D. VanGuilder, Kent E. Vrana, and Willard M. Freeman. “Twenty-Five Years of Quantitative PCR for Gene Expression Analysis”. In: *BioTechniques* 44.5 (Apr. 2008), pp. 619–626. ISSN: 0736-6205. DOI: 10.2144/000112776.

[95] Jianxin Wang et al. “A Fast Hierarchical Clustering Algorithm for Functional Modules Discovery in Protein Interaction Networks”. In: *IEEE/ACM Trans. Comput. Biol. Bioinformatics* 8.3 (May 2011), pp. 607–620. ISSN: 1545-5963. DOI: 10.1109/TCBB.2010.75.

[96] Rongquan Wang et al. “Predicting Overlapping Protein Complexes Based on Core-Attachment and a Local Modularity Structure”. eng. In: *BMC bioinformatics* 19.1 (Aug. 2018), p. 305. ISSN: 1471-2105. DOI: 10.1186/s12859-018-2309-9.

[97] D. C. Weaver, C. T. Workman, and G. D. Stormo. “Modeling Regulatory Networks with Weight Matrices”. In: *Biocomputing ’99.* WORLD SCIENTIFIC, Dec. 1998, pp. 112–123. ISBN: 978-981-02-3624-3. DOI: 10.1142/9789814447300_0011.

[98] S Wright. “The Roles of Mutation, Inbreeding, Crossbreeding, and Selection in Evolution”. In: *Proc 6th Int. Cong. Genet.* 1 (1932), pp. 356–366.

[99] Min Wu et al. “A Core-Attachment Based Method to Detect Protein Complexes in PPI Networks”. In: *BMC Bioinformatics* 10.1 (June 2009), p. 169. ISSN: 1471-2105. DOI: 10.1186/1471-2105-10-169.

[100] Tong Tong Wu and Kenneth Lange. “Coordinate Descent Algorithms for Lasso Penalized Regression”. In: *The Annals of Applied Statistics* 2.1 (Mar. 2008), pp. 224–244. ISSN: 1932-6157. DOI: 10.1214/07-AOAS147. arXiv: 0803.3876. URL: http://arxiv.org/abs/0803.3876 (visited on 07/14/2019).

[101] J. Xie, B. K. Szymanski, and X. Liu. “SLPA: Uncovering Overlapping Communities in Social Networks via a Speaker-Listener Interaction Dynamic Process”. In: *2011 IEEE 11th International Conference on Data Mining Workshops.* Dec. 2011, pp. 344–349. DOI: 10.1109/ICDMW.2011.154.

[102] M Yuan and Y Lin. “Model Selection and Estimation in Regression with Grouped Variables”. In: *J R Stat Soc Series B Stat Methodol* (2006).

[103] Zehua Zhang et al. “Detecting Complexes from Edge-Weighted PPI Networks via Genes Expression Analysis”. In: *BMC Systems Biology* 12.4 (Apr. 2018), p. 40. ISSN: 1752-0509. DOI: `10.1186/s12918-018-0565-y`.

# Appendix A

# Learning Interactions

## A.1 Number of epistatic gene pairs

For $n = 10 \times p = 1000$ siRNAs, 87% of the $\binom{p}{2} = 4950$ gene pairs are simultaneously perturbed by at least one siRNA.

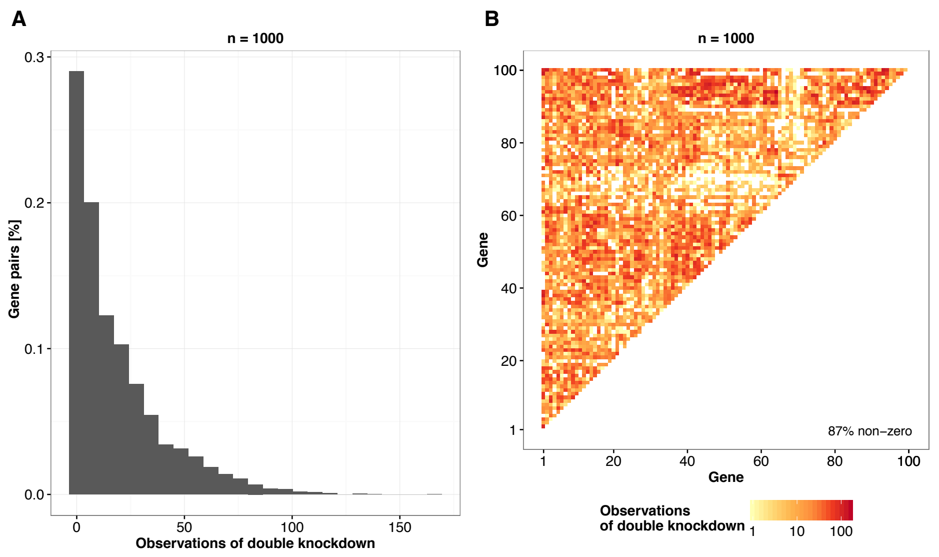
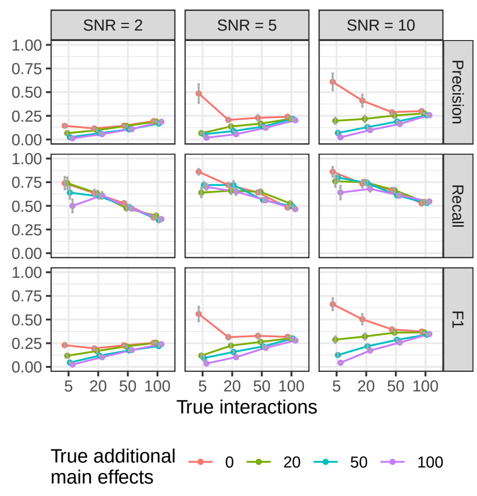


Figure A.1: Simulation of perturbation matrices for $n = 1000$ siRNAs and $p = 100$ genes based on four commercial genome-wide siRNA libraries from Qiagen. (**A**) The number of times each pair of genes is simultaneously perturbed in the simulated matrix. (**B**) Heat map of the number of simultaneous perturbations for each gene pair. Darker colour indicates higher numbers of observations. 87% of the $\binom{p}{2}$ pairs are simultaneously perturbed at least once.

An increase in the number of pairs of genes $(i, j) : \beta_{i,j} > 0$, i.e. pairs of genes with true conditional epistasis greater than zero, generally leads to an increase in precision and decrease in recall which results in a subtle increase in F1 when searching with `glinternet` (Figure A.2a). The only exception being when there are no additional

main effects, in which case interactions are more reliably found from among a small set (between 5 and 20 depending on the SNR) than a large one (50 or more). When we select estimates $\hat{\beta}_{i,j}$ with a magnitude significantly different from zero (Figure A.2b), we observe a more than 3-fold increase of precision but steeper decrease of recall for increasing numbers of pairs of genes with true conditional epistasis. This results in approximately 2-fold increase of the F1 measure, which in addition shows a weaker dependency on the number of gene pairs with true conditional epistasis. With an increasing number of additional main effects, the performance generally decreases. The effect is more subtle for high numbers of true epistatic gene pairs, both with and without selecting $\hat{\beta}_{i,j}$ significantly different from zero. As expected, higher SNRs leads to better performance, where this effect is stronger when we perform the significance test. The trade-off between precision and recall resulting from the significance test is shown in Figure A.4a. The increase in precision and decrease of recall is stronger for higher number of true epistatic gene pairs. For small numbers of true epistatic gene pairs (5, 20) we observe a dependency of the strength of increase of precision and decrease of recall to the number of additional main effects. Overall, the ratio of increase in precision and decrease of recall is approximately 3, suggesting that the test in general led to an increase in performance. Figures without this test may be found in Section A.4.
It should be noted that the expected precision of random guessing of interactions is $\frac{q}{p(p-1)}$. This is at most $\approx 1\%$, when $q = 100, p = 100$, as in our simulations.

(a)



(b)

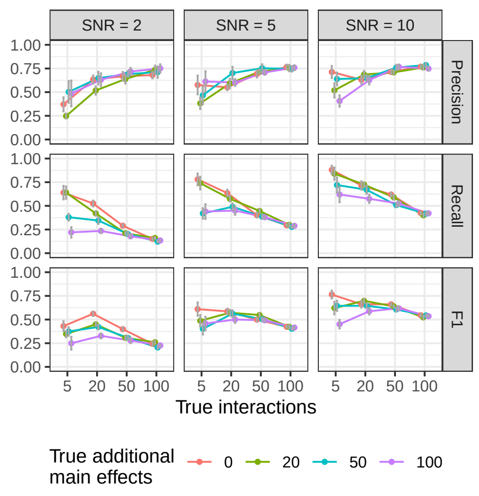
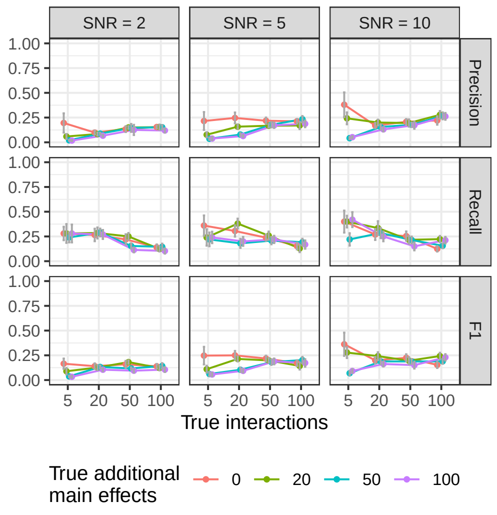


Figure A.2: Identification of epistasis for increasing numbers of true interactions using `glinternet`. Panel rows show precision, recall, and the F1 measure and panel columns depict results for signal-to-noise ratios (SNR) 2, 5, and 10. Colour indicates the number of additional main effects not overlapping with the set of interacting genes. (a) Results for all conditional epistasis $\beta_{i,j} > 0$; (b) Results for the subset of conditional epistasis $\beta_{i,j}$ that significantly deviate from zero (q-value $< 0.05$).

Using the same number of genes and main effects and searching with `xyz`, we see similar precision, albeit with significantly lower recall (Figure A.3). As with `glinternet`, performance improved with higher SNRs. Selecting estimates that significantly deviate from zero (q-value < 0.05) results in as much as a 2-fold improvement in precision in the best case, however improvements are generally smaller with `xyz` than with `glinternet`. In this case, the effect on recall is minimal, the trade-off is shown in Figure A.4b.

(a)



(b)

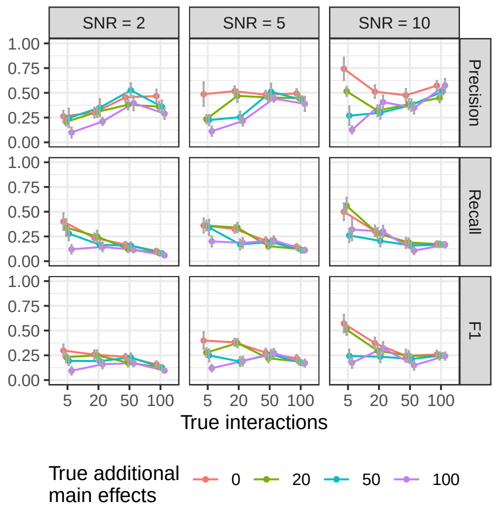
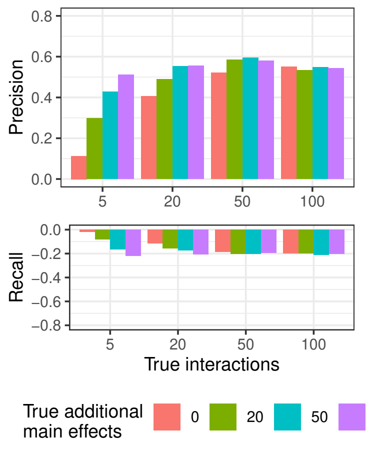


Figure A.3: Results for `xyz` as in Figure A.2. Note that this format is reused in all such figures. (a) Without significance test. (b) With significance test.

(a)



(b)

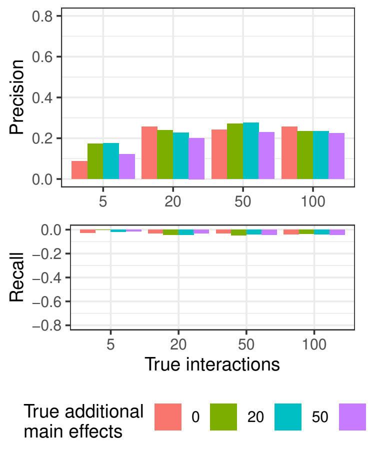


Figure A.4: Trade-off between precision and recall for selecting the subset of interactions significantly deviating from zero versus all interactions. Top and bottom panels depict gain of precision and loss of recall, respectively. (a) `glinternet`; (b) `xyz`.

## A.2 Magnitude

Comparing the estimated magnitude of epistasis to the ground truth, we find the `glinternet` results typically deviate less than 5%, and are only larger with a large number of true interactions, and a low signal to noise ratio. Using `xyz` we can see some significant variation in accuracy. The deviation is, however, typically below 10%.

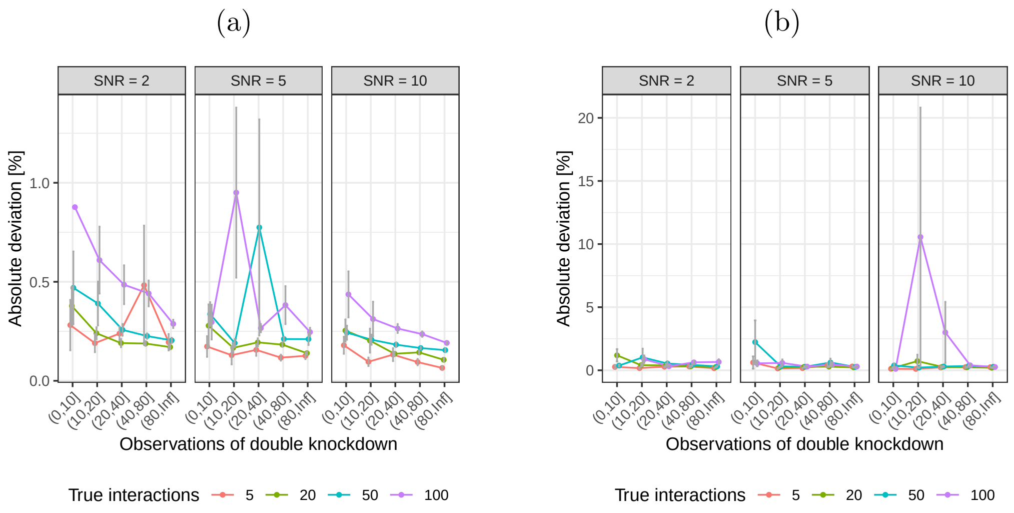


Figure A.5: Concordance between the magnitude of true and estimated epistasis. The fraction of incorrectly identified signs between true and estimated epistasis for (a) `glinternet` and (b) `xyz`. Results are for the subset of interactions that significantly deviate from zero (q-value $< 0.05$).

## A.3 Number of `xyz` Projections

To ensure the correct `xyz` parameters are chosen, we compare precision, recall, and F1 for varying numbers of projections. Fixing the signal to noise ration to $SNR = 5$, and using the same parameters as the main $p = 1000$ simulations above, we run `xyz` with $L = 10, 100$, and $1000$.

While there is a clear advantage to running at least $L = \sqrt(n) = 100$ projections, there are no significant gains in overall performance, as indicated by $F1$, beyond that. In fact, we can see in Figure A.6c that increasing the number of projections beyond that merely reduces the number of interactions returned, without improving accuracy.

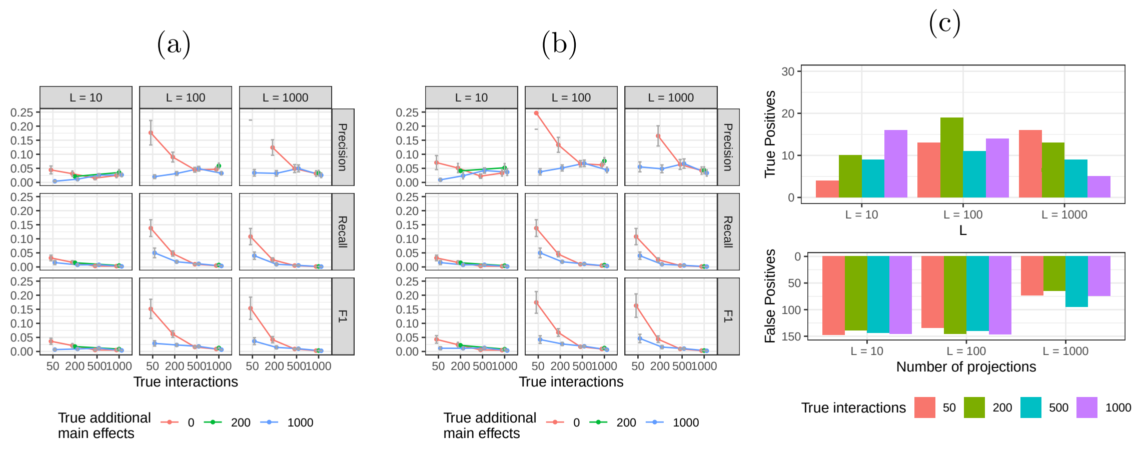


Figure A.6: Precision, recall, and F1 as a result of increasing the number of projections. We use $p = 1000$ genes with a signal to noise ratio of five. A.6a: Results considering all identified conditional epistasis. A.6b: Results considering only the subset of conditional epistasis that significantly deviate from zero. A.6c: Number of interactions reported, note that the scale above differs from the one below for readability.

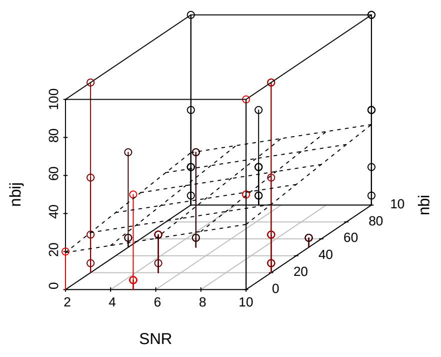
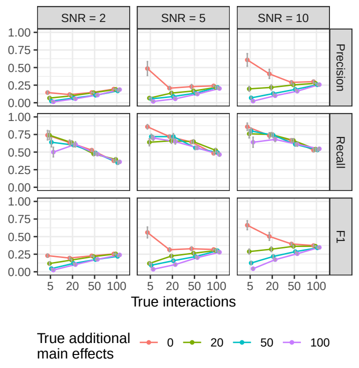


Figure A.7: Distribution of `xyz` failures.

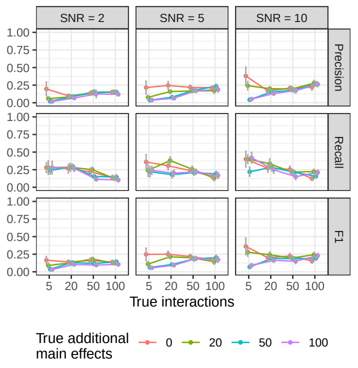
## A.4 Results without Significance Test

The results used above are using R's `lm` linear regression, including only those with significant q-values ($q < 0.05$). Here we perform the same tests with all found effects included, significant or otherwise, or comparison.

(a)



(b)



(c)

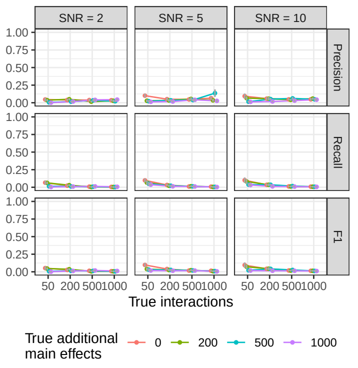
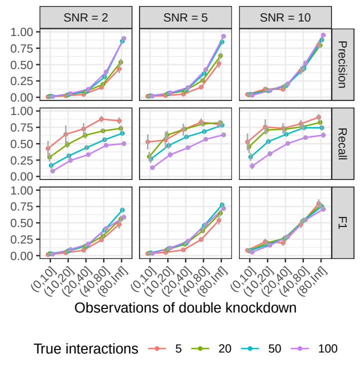
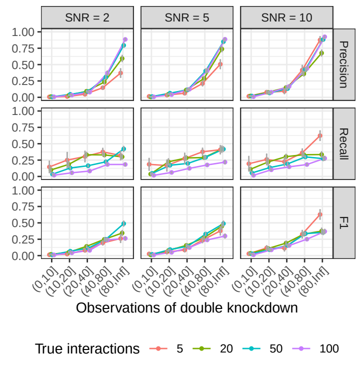


Figure A.8: Precision, recall, and F1 performance measures for `xyz`. A.8a: Results for all identified epistasis on $p = 100$, $n = 100$ simulation using `glinternet`. A.8b: Results for all identified epistasis on $p = 100$, $n = 100$ simulation using `xyz`. A.8c: Results for all identified epistasis on $p = 1000$, $n = 10000$ simulation using `xyz`.

(a)



(b)



(c)

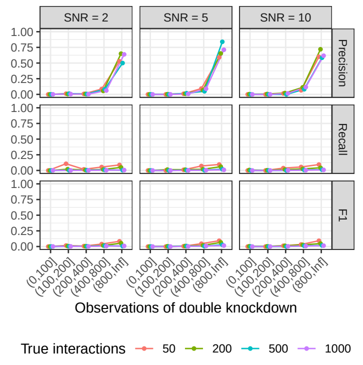


Figure A.9: Identification of epistasis for increasing numbers of observations of the pairwise double knockdown. Results are for all identified conditional epistasis $\beta_{i,j} > 0$. (a) Results using `glinternet`. (b) Results using `xyz` on small $(p = 100, n = 1000)$ simulations. (c) Results using `xyz` on large $(p = 1000, n = 10000)$ simulations.

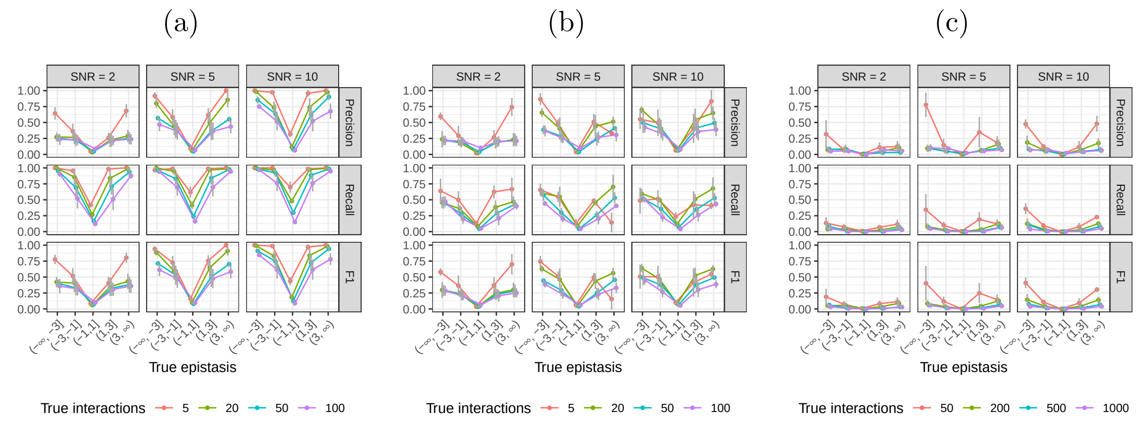


Figure A.10: Identification of epistasis for varying effect size. Results are for all identified conditional epistasis $\beta_{i,j} > 0$. (b) Small ($p = 100, n = 1000$) simulations, using `glinternet`. (b) Small ($p = 100, n = 1000$) simulations, using `xyz`. (c) Large $p = 1000$, $n = 10000$ simulations, using `xyz`.

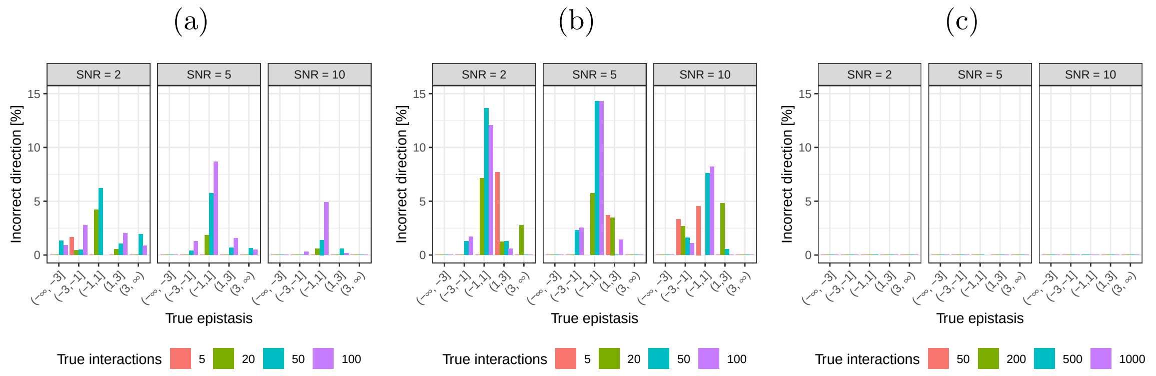


Figure A.11: Identification of epistasis for varying effect size. Results are for all identified conditional epistasis $\beta_{i,j} > 0$. (b) Small ($p = 100, n = 1000$) simulations, using `glinternet`. (b) Small ($p = 100, n = 1000$) simulations, using `xyz`. (c) Large $p = 1000$, $n = 10000$ simulations, using `xyz`. Note that in this test there are no incorrect results

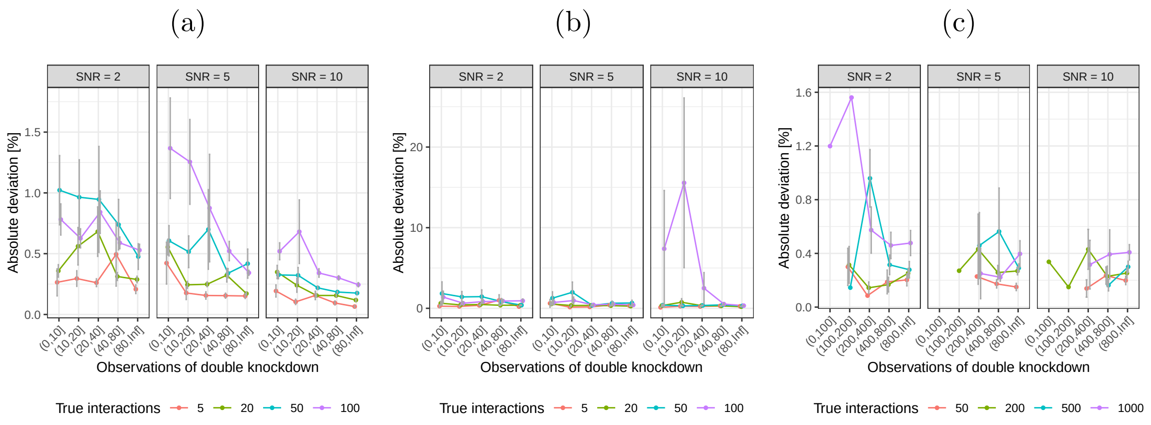


Figure A.12: Concordance between the magnitude of true and estimated epistasis. Results are for all identified conditional epistasis $\beta_{i,j} > 0$. (a) Small ($p = 100, n = 1000$) simulations, using `glinternet`. (b) Small ($p = 100, n = 1000$) simulations, using `xyz`. (c) Large $p = 1000,\ n = 10000$ simulations, using `xyz`.

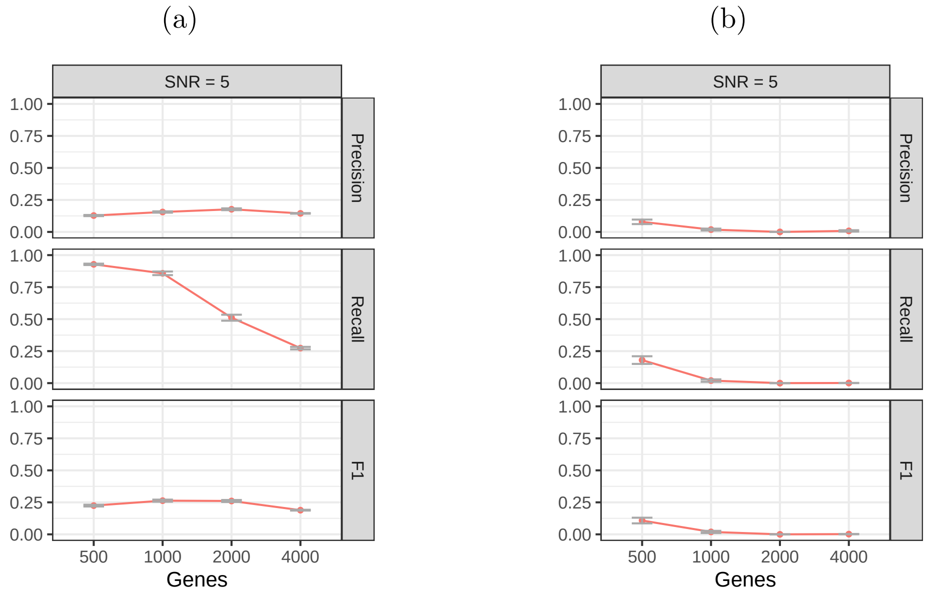


Figure A.13: The performance of (a) `glinternet` and (b) `xyz` on increasingly large data sets. Results are for all identified conditional epistasis $\beta_{i,j} > 0$.