Seidr Documentation

Release 0.14.0

Bastian Schiffthaler, Alonso Serrano, Nicolas Delhomme

Dec 04, 2020

Getting Started

Seiðr is a toolkit to create crowd networks. We provide fast implementations of several highly regarded algorithms as well as utility programs to create and explore crowd networks.

If you have any questions, contact me at: bastian.schiffthaler@umu.se

CHAPTER 1

Building Seidr

1.1 Supported OSs

`seidr` should build fine on most Linux distributions.

Test builds of `seidr` are created on Ubuntu 18.04 and Fedora 31. It is possible to build on Mac OS X (with some effort). Microsoft Windows is currently not supported.

1.2 Basic Build

Currently, `seidr` has the following dependencies (examplary dnf packages on Fedora):

- `gcc`
- `gcc-c++`
- `gcc-gfortran`
- `cmake`
- `git`
- `boost-devel`
- `glpk-devel` or `coin-or-Clp-devel` (see *A note on CLP and GLPK*)
- `armadillo-devel`
- `zlib-devel`

Once the dependencies are satisfied, build with:

```
git clone --recursive https://github.com/bschiffthaler/seidr
cd seidr
mkdir build
```

(continues on next page)

(continued from previous page)

```
cmake -DCMAKE_BUILD_TYPE=Release ..
make
```

If you have multiple CPU cores, run make as `make -j <ncpus>` to speed up building.

1.3 Building with MPI

If you have access to a compute cluster, you might want to build `seidr` with MPI support. If you have the MPI libraries installed (e.g.: `openmpi-devel` on Fedora) add:

```
cmake -DSEIDR_WITH_MPI=ON ..
```

You will need in addition these packages:

- `openmpi-devel`

to the CMake build options. This is only beneficial if you want to run multi-node jobs, as otherwise the overhead from MPI will decrease overall performance on a single node.

1.4 Building Parallel STL (PSTL)

If you have Intel TBB and PSTL availble, you can build `seidr` with support for parallel STL algorithms, which can speed up some operations. To do that, add:

```
cmake -DSEIDR_PSTL=ON ..
```

to the CMake build options.

1.5 A note on CLP and GLPK

The `narromi` algorithm uses linear programming routines, which in `seidr` is implemented via either GLPK or CLP backends. GLPK is widely available, but not safe to use in an OpenMP context, you will therefore be limited to a single OMP thread. CLP is safer, but packages are less widely available (you might need to build form source). If you want to build `seidr` with the CLP backend add:

```
cmake -DNARROMI_USE_CLP=ON ..
```

CHAPTER 2

Getting Started

2.1 Tutorial Data

The tutorial data is a set of 500 salmon pseudo-alignments targeting *Saccharomyces cerevisiae*. They originate from public SRA data. In fact the files names are simply their SRA accessions. In this tutorial, we will go through a simple pre-processing step and then create individual and aggregate networks. First, let's get the data:

```
mkdir seidr_tutorial
cd seidr_tutorial
wget https://bschiffthaler.s3-eu-west-1.amazonaws.com/SeidrPublic/seidr_tutorial.tar.
↪gz
tar -xvf seidr_tutorial.tar.gz
```

2.2 Pre-processing

Data pre-processing is done in R. We'll use tximport to load the data into R, and then use DESeq2 to variance-stabilize.

```
library(tximport)
library(DESeq2)
library(readr)

# This load the mapping of genes to transcripts into R so that tximport can
# summarize counts
tx2g <- read_tsv('seidr_tutorial/tx2gene.tsv',
                 col_names=c('Transcript', 'Gene'))

# Now let's find all our count files and load them into R using tximport
input_files <- dir('seidr_tutorial', pattern='*_quant.sf$', full.names=TRUE)
txi <- tximport(input_files, 'salmon', tx2gene=tx2g)

# In order to make a DESeq2 data set, we need some metadata. For now we'll
```

(continues on next page)

(continued from previous page)

```
# just use some dummy data
dummy_meta <- data.frame(N = seq_along(input_files))
dds <- DESeqDataSetFromTximport(txi, dummy_meta, ~1)

# Now we run variance stabilization and get the stabilized data as a matrix. If
# you have good metadata, you can use the experimental design in the DESeqDataSet
# and set blind=FALSE here
vsd <- varianceStabilizingTransformation(dds, blind=TRUE)
vst <- t(assay(vsd))

# Genes that do not vary at all create problems down the line, so it's better
# to drop them
vars <- apply(vst, 2, var)
filt_id <- which(is.finite(vars))
vst <- vst[, filt_id]

# Let's also center samples around their median, which has been shown to
# improve reconstruction accuracy
medians <- apply(vst, 1, median)
vst <- sweep(vst, MARGIN=1, FUN='-', STATS=medians)

# MASS's write.matrix function is a bit faster and better suited for our task
# compared to write.table. Don't forgot to unname(), otherwise you will have
# column headers in the output
MASS::write.matrix(x=unname(vst), sep='\t', file='expression.tsv')

# Finally, let's write the column headers (== gene names) as a text file
write(colnames(vst), file="genes.txt")
```

2.3 All versus all networks

2.3.1 Sub-setting the data

In this section, we will create an all vs. all comparison, meaning we will estimate connectivity of all genes to all other genes. This approach is the most resource demanding, so we'll create a smaller subset of the tutorial data first. We'll take 100 samples and the first 1000 genes. Our output network will therefore be $\frac{1000 \cdot 999}{2}$.

```
tail -n 100 expression.tsv | cut -f 1-1000 > expression_sub.tsv
head -n 1000 genes.txt > genes_sub.txt
```

2.3.2 Network inference

Now that we have a data subset, we can get started with the inference. In this step, we'll create 13 different all vs. all networks using algorithms that seidr ships with. If you have any inference algorithm you would like to include that is not yet implemented in seidr, you can run that as well, but make sure its output is in a format seidr can import (*Importing text based formats into Seidr*). Even though this is a sub-set, you'll probably need to set aside an hour for the inference. If you want a quicker run, you can leave out `el-ensemble`. You'll see that we use the `--scale` option a number of times. This instructs seidr to perform feature scaling on the data, which, in general, will improve the results.:

```
# fast
correlation -m pearson -i expression_sub.tsv -g genes_sub.txt --scale
correlation -m spearman -i expression_sub.tsv -g genes_sub.txt
pcor -i expression_sub.tsv -g genes_sub.txt --scale

# medium
mi -m RAW -i expression_sub.tsv -g genes_sub.txt -o mi_scores.tsv
mi -m CLR -i expression_sub.tsv -g genes_sub.txt -M mi_scores.tsv -o clr_scores.tsv
mi -m ARACNE -i expression_sub.tsv -g genes_sub.txt -M mi_scores.tsv -o aracne_scores.
↪tsv

# slow
narromi -m interior-point -i expression_sub.tsv -g genes_sub.txt -o narromi_scores.tsv
plsnet -i expression_sub.tsv -g genes_sub.txt -o plsnet_scores.tsv --scale
llr-ensemble -i expression_sub.tsv -g genes_sub.txt -o llr_scores.tsv --scale
svm-ensemble -k POLY -i expression_sub.tsv -g genes_sub.txt -o svm_scores.tsv --scale
genie3 -i expression_sub.tsv -g genes_sub.txt -o genie3_scores.tsv --scale
tigress -i expression_sub.tsv -g genes_sub.txt -o tigress_scores.tsv --scale

# very slow
el-ensemble -i expression_sub.tsv -g genes_sub.txt -o elnet_scores.tsv --scale
```

2.3.3 Network ranking

Different inference algorithms output networks with different metrics for edge weights. A correlation network, will assign scores anywhere in $[-1, ..., 1]$, whereas mututal information is in $[0, ..., N]$, and many of the regression algorithms in $[0, ..., 1]$. We can therefore not just sum the weights to get a final, community network. In order to do that, we want to convert the scores to ranks. The command `seidr import` takes care of that. Some useful options:

- `-A`: This option computes the rank on the **absolute** value of the score, so -1 and +1 would get the same rank.
- `-r`: This option indicates that **higher scores are better**. A score of 1 would get a lower (== better) rank than a score of 0.5.
- `-u`: This option creates an undirected network. We use this in algorithms where we know the output is symmetric (A->B and B->A are the same), but only have the lower triangular matrix. Examples are all correlation and mutual information based methods.
- `-z`: This option drops edges with a score of 0. By default we keep all edges, but this will create sparser networks for methods that output 0-valued edges.

```
seidr import -A -r -u -n PEARSON -o person_scores.sf -F lm -i pearson_scores.tsv -g␣
↪genes_sub.txt
seidr import -A -r -u -n SPEARMAN -o spearman_scores.sf -F lm -i spearman_scores.tsv -
↪g genes_sub.txt
seidr import -A -r -u -n PCOR -o pcor_scores.sf -F lm -i pcor_scores.tsv -g genes_sub.
↪txt

seidr import -r -u -n MI -o mi_scores.sf -F lm -i mi_scores.tsv -g genes_sub.txt
seidr import -r -u -z -n CLR -o clr_scores.sf -F lm -i clr_scores.tsv -g genes_sub.txt
seidr import -r -u -z -n ARACNE -o aracne_scores.sf -F lm -i aracne_scores.tsv -g␣
↪genes_sub.txt

seidr import -r -z -n NARROMI -o narromi_scores.sf -F m -i narromi_scores.tsv -g␣
↪genes_sub.txt
seidr import -r -z -n PLSNET -o plsnet_scores.sf -F m -i plsnet_scores.tsv -g genes_
↪sub.txt
```

(continues on next page)

(continued from previous page)

```
seidr import -r -z -n LLR -o llr_scores.sf -F m -i llr_scores.tsv -g genes_sub.txt
seidr import -r -z -n SVM -o svm_scores.sf -F m -i svm_scores.tsv -g genes_sub.txt
seidr import -r -z -n GENIE3 -o genie3_scores.sf -F m -i genie3_scores.tsv -g genes_
↪sub.txt
seidr import -r -z -n TIGRESS -o tigress_scores.sf -F m -i tigress_scores.tsv -g␣
↪genes_sub.txt
seidr import -r -z -n ELNET -o elnet_scores.sf -F m -i elnet_scores.tsv -g genes_sub.
↪txt
```

2.3.4 Aggregating

Aggregating refers to the construction of a community network from the individual networks created before. Note that there are several aggregation methods available. We will use the "Inverse Rank Product" method described in [Zhong2014].

```
seidr aggregate -m irp aracne_scores.sf clr_scores.sf elnet_scores.sf genie3_scores.
↪sf llr_scores.sf mi_scores.sf narromi_scores.sf pcor_scores.sf person_scores.sf␣
↪plsnet_scores.sf spearman_scores.sf svm_scores.sf tigress_scores.sf
```

This creates a community network of all the 1000 genes in our sample data. If you don't want to learn how you can create a network for a group of genes (e.g. only transcription factors), jump right to *Post processing*.

2.3.5 Taking a look at the final network

We can have a look at the top three edges in the network:

```
seidr top -n 3 aggregated.sf | column -t
```

```
YDL039C    YDL037C    Undirected  0.777779;83  14.1252;1   0.511;433  3.61084;7   0.
↪138;61223.5  0.777779;204  0.709272;39   0.0787986;93     0.837201;940  1.5595;25  ␣
↪   0.780375;1512  nan;nan        1;1.5       0.949186;3
YDL025W-A  YBL006W-A  Undirected  1.05879;2    8.3004;29   0.543;4    3.30464;21  0.
↪496;2775     1.05879;2     0.372718;529  0.0172536;31848  0.909562;219  0.654752;
↪1758  0.847192;364   0.463;4415     0.8677;144  0.967855;2
YAL037C-B  YCR013C    Directed    1.00539;7    9.08174;14  0.519;168  2.82549;92  0.
↪517;407      1.00539;7     0.777048;28   0.032911;3601    0.928263;137  0.991901;
↪297   0.846061;373   0.204;12028.5  0.99325;16  1;1
```

Most of these are from dubious ORFs (which should have maybe been filtered beforehand). The one that is not, is definitely a good result, YDL039C and YDL037C as both these genes form the IMI1 protein.

2.4 Creating targeted networks

Sometimes, we are not interested in the interactions of all genes, we just want to know what our genes of interest look like in the network. We can then run seidr in targeted mode, which will compute only what's necessary to understand that particular group of genes. The slowest of the bunch will probably be the mutual information based algorithms CLR and ARACNe, since they are context dependent and the full mutual information matrix needs to be computed first.

2.4.1 Making targets

Since we need some targets to look at, I select a single transcription factor FZF1, and store the gene identifier in a file.

```
echo "YGL254W" > FZF1.txt
```

2.4.2 Inferring sub-networks

The network inference step is nearly the same, but now we use the full expression set (all ~6500 genes and 500 samples) as well as the `FZF1.txt` targets file.

```
# fast
correlation -t FZF1.txt -m pearson -i expression.tsv -g genes.txt --scale -o pearson_
↪fzf1_scores.tsv
correlation -t FZF1.txt -m spearman -i expression.tsv -g genes.txt -o spearman_fzf1_
↪scores.tsv
pcor -t FZF1.txt -i expression.tsv -g genes.txt --scale -o pcor_fzf1_scores.tsv

# medium
mi -t FZF1.txt -m RAW -i expression.tsv -g genes.txt -M mi_full_scores.tsv -o mi_fzf1_
↪scores.tsv
mi -t FZF1.txt -m CLR -i expression.tsv -g genes.txt -M mi_full_scores.tsv -o clr_
↪fzf1_scores.tsv
mi -t FZF1.txt -m ARACNE -i expression.tsv -g genes.txt -M mi_full_scores.tsv -o␣
↪aracne_fzf1_scores.tsv

# slow
narromi -t FZF1.txt -m interior-point -i expression.tsv -g genes.txt -o narromi_fzf1_
↪scores.tsv
plsnet -t FZF1.txt -i expression.tsv -g genes.txt -o plsnet_fzf1_scores.tsv --scale
llr-ensemble -t FZF1.txt -i expression.tsv -g genes.txt -o llr_fzf1_scores.tsv --scale
svm-ensemble -t FZF1.txt -k POLY -i expression.tsv -g genes.txt -o svm_fzf1_scores.
↪tsv --scale
genie3 -t FZF1.txt -i expression.tsv -g genes.txt -o genie3_fzf1_scores.tsv --scale
tigress -t FZF1.txt -i expression.tsv -g genes.txt -o tigress_fzf1_scores.tsv --scale

el-ensemble -t FZF1.txt -i expression.tsv -g genes.txt -o elnet_fzf1_scores.tsv --
↪scale
```

2.4.3 Importing

Targeted mode outputs results in edge list format, so all out imports now contain `-F el` instead of `-F lm` or `-F m`.

```
seidr import -A -r -u -n PEARSON -o person_fzf1_scores.sf -F el -i pearson_fzf1_
↪scores.tsv -g genes.txt
seidr import -A -r -u -n SPEARMAN -o spearman_fzf1_scores.sf -F el -i spearman_fzf1_
↪scores.tsv -g genes.txt
seidr import -A -r -u -n PCOR -o pcor_fzf1_scores.sf -F el -i pcor_fzf1_scores.tsv -g␣
↪genes.txt

seidr import -r -u -n MI -o mi_fzf1_scores.sf -F el -i mi_fzf1_scores.tsv -g genes.txt
seidr import -r -u -z -n CLR -o clr_fzf1_scores.sf -F el -i clr_fzf1_scores.tsv -g␣
↪genes.txt
seidr import -r -u -z -n ARACNE -o aracne_fzf1_scores.sf -F el -i aracne_fzf1_scores.
↪tsv -g genes.txt
```

(continues on next page)

(continued from previous page)

```
seidr import -r -z -n NARROMI -o narromi_fzf1_scores.sf -F el -i narromi_fzf1_scores.
↪tsv -g genes.txt
seidr import -r -z -n PLSNET -o plsnet_fzf1_scores.sf -F el -i plsnet_fzf1_scores.tsv␣
↪-g genes.txt
seidr import -r -z -n LLR -o llr_fzf1_scores.sf -F el -i llr_fzf1_scores.tsv -g genes.
↪txt
seidr import -r -z -n SVM -o svm_fzf1_scores.sf -F el -i svm_fzf1_scores.tsv -g genes.
↪txt
seidr import -r -z -n GENIE3 -o genie3_fzf1_scores.sf -F el -i genie3_fzf1_scores.tsv␣
↪-g genes.txt
seidr import -r -z -n TIGRESS -o tigress_fzf1_scores.sf -F el -i tigress_fzf1_scores.
↪tsv -g genes.txt
seidr import -r -z -n ELNET -o elnet_fzf1_scores.sf -F el -i elnet_fzf1_scores.tsv -g␣
↪genes.txt
```

2.4.4 Aggregating

This is exactly the same as for the full network.

```
seidr aggregate -m irp -o aggregated_fzf1.sf aracne_fzf1_scores.sf clr_fzf1_scores.sf␣
↪elnet_fzf1_scores.sf genie3_fzf1_scores.sf llr_fzf1_scores.sf mi_fzf1_scores.sf␣
↪narromi_fzf1_scores.sf pcor_fzf1_scores.sf person_fzf1_scores.sf plsnet_fzf1_scores.
↪sf spearman_fzf1_scores.sf svm_fzf1_scores.sf tigress_fzf1_scores.sf
```

2.4.5 Taking a look at the final network

Just as before, let's look at the top three connections of our TF.

```
seidr top -n 3 aggregated_fzf1.sf
```

```
YGL254W  YEL051W  Directed  nan;nan  3.11877;43     0.458;2    1.81032;4     0.004;
↪3226  0.387233;48    nan;nan       -0.00566415;1138  -0.57406;1    0.035306;28   -0.
↪462468;6  0.001;3050.5  0.74525;2  0.874875;3
YGL254W  YGL128C  Directed  nan;nan  0.886213;1151  0.443;4.5  0.266012;279  0.248;86␣
↪   0.265754;1222  0.0648351;43  0.010073;254      0.453637;131  0.0382971;21  0.
↪404141;49  0.207;210.5   0.19945;7  0.888152;2
YGL254W  YFL044C  Directed  nan;nan  3.24149;31     0.451;3    0.581017;113  0.352;17␣
↪   0.383645;52    0.1711;4      0.00935282;329    0.477231;78   0.0320321;45  0.
↪444953;11  0.041;1019    0.25265;4  1;1
```

The top connections are OTU1 (YFL044C), CWC23 (YGL128C), and VMA8 (YEL051W).

2.5 Post processing

2.5.1 Pruning noisy edges

In most cases, the community network will be fully dense, meaning every gene is connected to every other gene with a certain score. Many of these edges are just noise and we would like to prune them. [Coscia2017] have developed a smart approach to pruning noisy edges called "Network backboning". We can apply this to our community network as:

```
seidr backbone -F 1.28 aggregated.sf
```

2.5.2 Viewing edges in the network

The `seidr view` command offers an interface to query the seidr output. Let's look at a few edges.

```
seidr view --column-headers aggregated.bb.sf | head -n 3 | column -t
```

```
Source  Target  Type        ARACNE_score;ARACNE_rank  CLR_score;CLR_rank  ELNET_score;
↪ELNET_rank  GENIE3_score;GENIE3_rank  LLR_score;LLR_rank  MI_score;MI_rank  NARROMI_
↪score;NARROMI_rank  PCOR_score;PCOR_rank  PEARSON_score;PEARSON_rank  PLSNET_score;
↪PLSNET_rank  SPEARMAN_score;SPEARMAN_rank  SVM_score;SVM_rank  TIGRESS_score;
↪TIGRESS_rank  irp_score;irp_rank  NC_Score;NC_SDev;SEC;EBC
Q0017   Q0010   Undirected  nan;nan                    3.58054;6569        0.317;14818␣
↪            0.587286;20930            0.221;43021         0.255911;100939   0.
↪364063;574                0.0246169;10270       0.644044;14824              1.01683;
↪268               0.509318;33915               0.154;13896         0.17055;3855.5 ␣
↪            0.440763;1637       0.602226;0.375774;0.459647;130
Q0032   Q0010   Undirected  nan;nan                    2.38815;27858       0.269;18028␣
↪            0.905035;9646             0.138;61223.5       0.138116;379828   nan;nan␣
↪                    0.0481595;843         0.679379;10339              1.19476;122  ␣ ␣
↪            0.386693;81327               0.287;9567          0.0787;8143.5        ␣
↪      0.37659;3358        0.621852;0.388965;0.364875;218
```

2.5.3 Querying specific nodes or edges

If the seidr output is indexed with the `seidr index` command, we can query specific nodes and edges.

```
seidr index aggregated.bb.sf
# Node
seidr view -n YBR142W aggregated.bb.sf
# Edge
seidr view -n YBR142W:YDL063C aggregated.bb.sf
```

2.5.4 Graph and centrality statistics

Seidr can compute statistics on the entire graph and some node centrality measures. Before we do that, it's best to make sure we have no disconnected nodes in the graph, which we drop with:

```
seidr reheader aggregated.bb.sf
```

Then, we can use `seidr graphstats` to compute graph summary stats.

```
seidr graphstats aggregated.bb.sf
```

```
Number of Nodes:        974
Number of Edges:        4150
Number of Connected Components: 2
Global clustering coefficient:  0.338051
Scale free fit: 0.0555276
Average degree: 8.52156
```

(continues on next page)

(continued from previous page)

```
Average weighted degree:        3.71159
Network diameter:       4.00379
Average path length:    1.66305
```

Finally, we can compute node centrality statistics with `seidr stats`

```
seidr stats --exact aggregated.bb.sf
seidr view --centrality aggregated.bb.sf | sort -k2g | tail -n 5 | column -t
```

```
YBL006W-A  0.002748    1344   14.8438  0.140839     0.0339691
YBL039C    0.00279961  1322   14.4697  0.000898011  0.0339036
YBR142W    0.00282644  6460   14.3319  0.00113027   0.03388
YBL012C    0.00296141  10198  16.4288  0.156134     0.0342437
YAL045C    0.00306967  4364   17.8655  0.234332     0.0344952
```

CHAPTER 3

The Crowd Network Idea

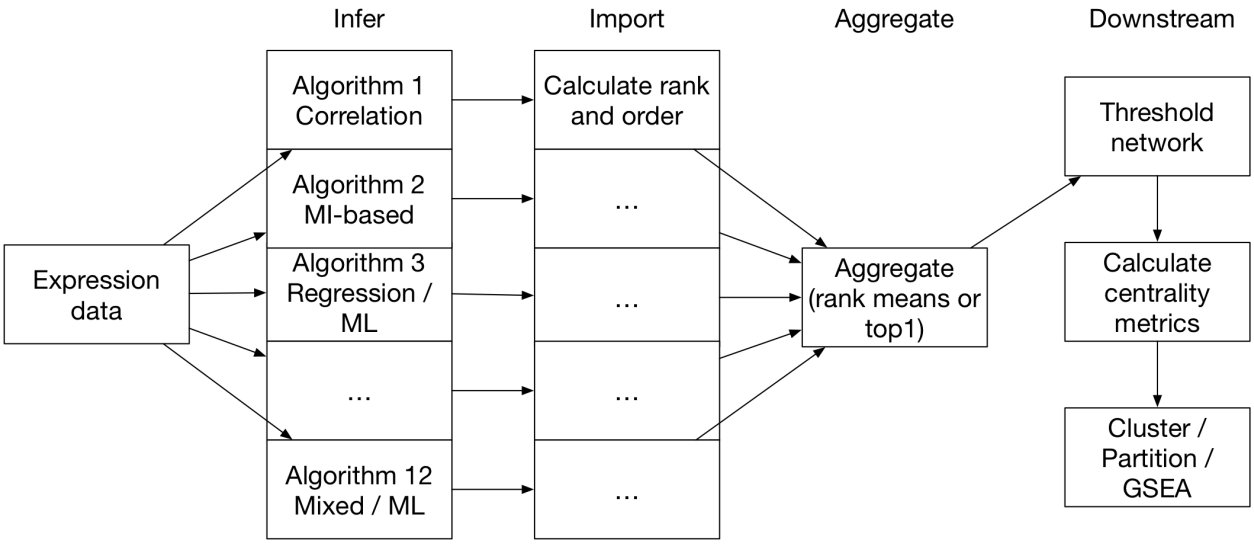
`Seidr` is a product of an idea presented in the DREAM 5 network challenge [Marbach2012]. In it, the authors show that gene regulatory network inference algorithms tend to suffer from biases towards specific interaction patters. They suggested a way to get around this by creating an aggregate of all the methods used in the study: a crowd network.

While the paper is widely cited, there is little software that attempts to integrate the findings. `Seidr` is an attempt to create a toolbox that simplifies the laborious effort of creating crowd networks.

3.1 The basic pipeline

A typical run of `seidr` has three steps:

- Infer: In the inference step, independent gene-gene networks are created by a multitude of algorithms.
- Import: In order to merge these networks, they are first sorted and ranked. To achieve this `seidr` uses its own file format: SeidrFiles (see *SeidrFiles*).
- Aggregate: Once all methods are ready, `seidr` can aggregate them to a crowd network.



In principle, any network can be input into `seidr`, as long as it was constructed under similar assumptions as all other networks. For example, it would be a bad idea to take a subset of genes and create a network, which is then aggregated with another subset using *different* genes. `Seidr` provides a number of algorithms as native applications written in C++:

Name	Published	Type	Orig. Lang.	Seidr Lang.	Orig. Parallel	Seidr Parallel
ANOVERENCE	[Kueffner2012]	ANOVA	C++	C++	No	No
ARACNE	[Margolin2006]	MI + DPI	C++	C++	Yes	Yes
CLR	[Faith2007] [Daub2004]	MI + CLR	MATLAB / C / C++	C++	No	Yes
Elastic Net ensemble	[Ruyssinck2014]	Elastic Net Regression	R (glmnet)	C++ (glmnet)	No	Yes
GENIE3	[Huynh-Thu2010]	Random Forest Regression	R (randomForest)	C++ (ranger)	No	Yes
NARROMI	[Zhang2013]	MI + Linear Programming	MATLAB	C++ (glpk)	No	Yes
Partial Correlation	[Schafer2005]	Correlation	R	C++	No	No
Pearson Correlation	NA	Correlation	NA	C++	No	No
PLSNET	[Guo2016]	PLS	MATLAB	C++	No	Yes
Spearman Correlation	NA	Correlation	NA	C++	No	No
SVM ensemble	[Ruyssinck2014]	SVM regression	R (libsvm) / C	C++ (libsvm or liblinear)	No	Yes
TIGRESS	[Haury2012]	LASSO Regression	MATLAB / R	C++ (glmnet)	No	Yes

3.2 Downstream

Once you have a network, you probably want to explore it. To that end we provide some utilities to investigate the networks and to prepare them for input into other software.

CHAPTER 4

Importing text based formats into `Seidr`

4.1 Introduction

Seidr works with its own binary file format, `SeidrFile` (see *SeidrFiles*). In order to convert text based formats to a `SeidrFile` we use the `seidr import` command. Format conversion is not the only thing `seidr import` does, it also ranks edge weights in the input files according to user parameters.

4.2 Import formats

`seidr import` currently supports three text based input formats.

Lower triangular matrix (`--format "lm"`)

A lower triangular matrix represents the lower half of a symmetric matrix. This is particularly useful for non directional inference algorithms. Take Pearson correlation as an example: If we correlate two vectors x, y, it does not matter if we check $x \sim y$ or $y \sim x$. It would therefore be a waste of space to store and a waste of computational resources to compute the second comparison. A lower trinagular matrix will have the result of each comparison exactly once:

```
    G1  G2  G3  G4  G5
G1
G2  0
G3  1   2
G5  3   4   5
G5  6   7   8   9
```

All cells with an index in the above matrix exist in the lower triangular. Note that the input is expected to be without headers, e.g.:

```
0
1   2
3   4   5
6   7   8   9
```

Matrix (`--format "m"`)

Opposed to the lower triangular, a matrix input is a square of all nodes vs all nodes (including self-self, which is ignored). This is the output of several machine learning algorithms which are non-symmetric (e.g. GENIE3, ELNET):

```
    G1  G2  G3  G4  G5
G1  0   1   2   3   4
G2  5   6   7   8   9
G3  10  11  12  13  14
G4  15  16  17  18  19
G5  20  21  22  23  24
```

Same as before, the input is expected to be without headers:

```
0   1   2   3   4
5   6   7   8   9
10  11  12  13  14
15  16  17  18  19
20  21  22  23  24
```

Edge lists (`--format "el"`)

Edge lists are simple TAB separated files, which describe one edge one line. They are relatively inefficient and store a lot of repetitive information, but they are very convenient for sparse networks and for humans to read:

```
G1  G2  0
G1  G3  1
G1  G4  2
G1  G5  3
G2  G3  4
G2  G4  5
G2  G5  6
G3  G4  7
G3  G5  8
G4  G5  9
```

ARACNE2 (`--format "ara"`)

While `seidr` can read output in the format of the original ARACNE2 output, the feature is not very well tested and should be considered experimental.

4.3 Importing your data

As a minimum, three arguments are required:

- `-i, --infile`: The input text file
- `-g, --genes` : A file containing all genes (nodes) in the input file
- `-F, --format`: The input format (`lm`, `m`, `el`)

Let's assume we have the lower triangular from before as output from our algorithm **lm.txt**:

```
0
1   2
3   4   5
6   7   8   9
```

We also have a file containg the names of nodes in the same order as the matrix. Note that for the lower triangular this file is assumed to be sorted as if it was column headers for the full (square) matrix. Generally, this is the same as column headers for the input data matrix of the algorithms **nodes.txt**:

```
G1
G2
G3
G4
G5
```

We can then run:

```
seidr import -i lm.txt -g nodes.txt -F lm
```

Once it finishes, we can view the output with:

```
$ seidr view elranks.sf
G2  G1  Directed  0;1
G3  G1  Directed  1;2
G3  G2  Directed  2;3
G4  G1  Directed  3;4
G4  G2  Directed  4;5
G4  G3  Directed  5;6
G5  G1  Directed  6;7
G5  G2  Directed  7;8
G5  G3  Directed  8;9
G5  G4  Directed  9;10
```

4.4 Adjusting import behaviour

Depending on the algorithm, the default behaviour might need to be adjusted. In the last example, we imported a lower triangular matrix, which by default creates all directed edges. In many cases, this might not be true as the lower triangular is likely to stem from a symmetric inference algorithm. The `-u, --undirected` option would do just that. Here are all modifiers:

- `-u, --undirected`: Forces all edges to be interpreted as undirected. Use when source data is from a symmetric method
- `-z, --drop-zero` : Regards edges with a score of 0 as missing. Use for sparse methods.
- `-r, --reverse` : Considers higher edge weights better. Use when a higher score means a more confident prediction. Most methods implemented in `seidr` work that way, e.g. an edge weight of 0.6 is better than one of 0.2. If you import data from an algorithm that computes e.g. P-values, you need to omit this flag, as lower P-values are better.
- `-A, --absolute` : Computes the ranking using absolute values. A good example for this is Pearson correlation. Both 1 and -1 are perfect correlations, but they tell different stories. We want to keep the sign intact, but give both edges the highest rank for aggregation, therefore we use this flag.

4.5 Naming imports

The last flag (`-n, --name`) lets you provide an internal name to the `SeidrFile` you are creating. Later, when you aggregate several `SeidrFiles` this will let you recognize the source of each score/rank column in the aggregated network.

4.6 A note on parallelism

If `seidr` was compiled with a compiler that supports OpenMP, `seidr import` will carry out some steps in parallel. You can control how many CPUs it should use with the `OMP_NUM_THREAD` environment variable. If you would like to turn multithreading for for example:

```
OMP_NUM_THREAD=1 seidr import -f lm.txt -g nodes.txt -F lm ...
```

CHAPTER 5

Aggregating networks into a crowd network

5.1 Introduction

Given a number of networks in `SeidrFile` format, seidr can aggregate those into a crowd network. The basic syntax is:

```
seidr aggregate <SeidrFile> <SeidrFile> ...
```

There are currently four methods of aggregation implemented:

- `-m borda`: This will ouput a mean of ranks.
- `-m top1`: This will ouput the edge with the highest score (==lowest rank) of all methods
- `-m top2`: This will ouput the middle of the two highest scores (==lowest ranks) of all methods
- `-m irp`: This will calculate the inverse rank product.

From a real example:

```
seidr aggregate -m irp ../elnet/elnet_scores.sf ../narromi/narromi_scores.sf ../
↪pearson/pearson_scores.sf ../spearman/spearman_scores.sf ../plsnet/plsnet_scores.sf␣
↪../aracne/aracne_scores.sf ../tigress/tigress_scores.sf ../clr/clr_scores.sf ../
↪genenet/genenet_scores.sf ../svm/svm_scores.sf ../llr/llr_scores.sf ../genie3/
↪genie3_scores.sf ../anova/anova_scores.sf
```

Without specifying an output file, this will create a file `aggregated.sf` in the current working directory. Each column after the third (excluding the supplementary) column stores the score and rank for each edge (if present) in all aggregated methods. Converted to text (with `seidr view`) the file looks like this:

```
Source  Target  Type  ELNET_score;ELNET_rank  Narromi_score;Narromi_rank  Pearson_
↪score;Pearson_rank  Spearman_score;Spearman_rank  PLSNET_score;PLSNET_rank  ARACNE_
↪score;ARACNE_rank  TIGRESS_score;TIGRESS_rank  CLR_score;CLR_rank  PCor_score;PCor_
↪rank  SVM_score;SVM_rank  LLR_score;LLR_rank  GENIE3_score;GENIE3_rank  ANOVA_score;
↪ANOVA_rank  irp_score;irp_rank
G2  G1  Undirected  0.004;334084  0.0128741;202752  -0.159435;202751  -0.00225177;1.
↪32058e+06 1.07712e-05;360264nan;nan nan;nan 1.87357;106802  -0.018736;[illegible]
↪26168 0.244;37455.5 0.0904447;42007 0.288087;1.30856e+06  0.176275;129253
```

(continues on next page)

(continued from previous page)

```
G3  G1  Undirected  0.334;22729.5 0.0381324;38394 -0.270978;44973 -0.214385;48864 3.
↪2165e-05;61265  nan;nan 0.0028;78346.5  2.27349;70552.5 -0.021059;184389  0.077;
↪91342.5 0.203;48670.5 0.215094;12249  0.388856;608154 0.299126;27713
```

We note that the final column stores the score of the aggregated network (IRP method). For all future purposes, this is the representative score unless otherwise specified.

CHAPTER 6

Estimating a hard threshold for a given seidr network

Note that generally ‘‘seidr backbone‘‘ is preferred to this approach. See *Calculating a Network Backbone*

Post aggregation, if any network in the input dataset was fully dense (i.e. having a score for each possible link in the network) the aggregated network will also be fully dense. The vast majority of the edges in the network will be noise, therefore we would like to find a cutoff that represents most of the signal being kept, and most of the noise trimmed away.

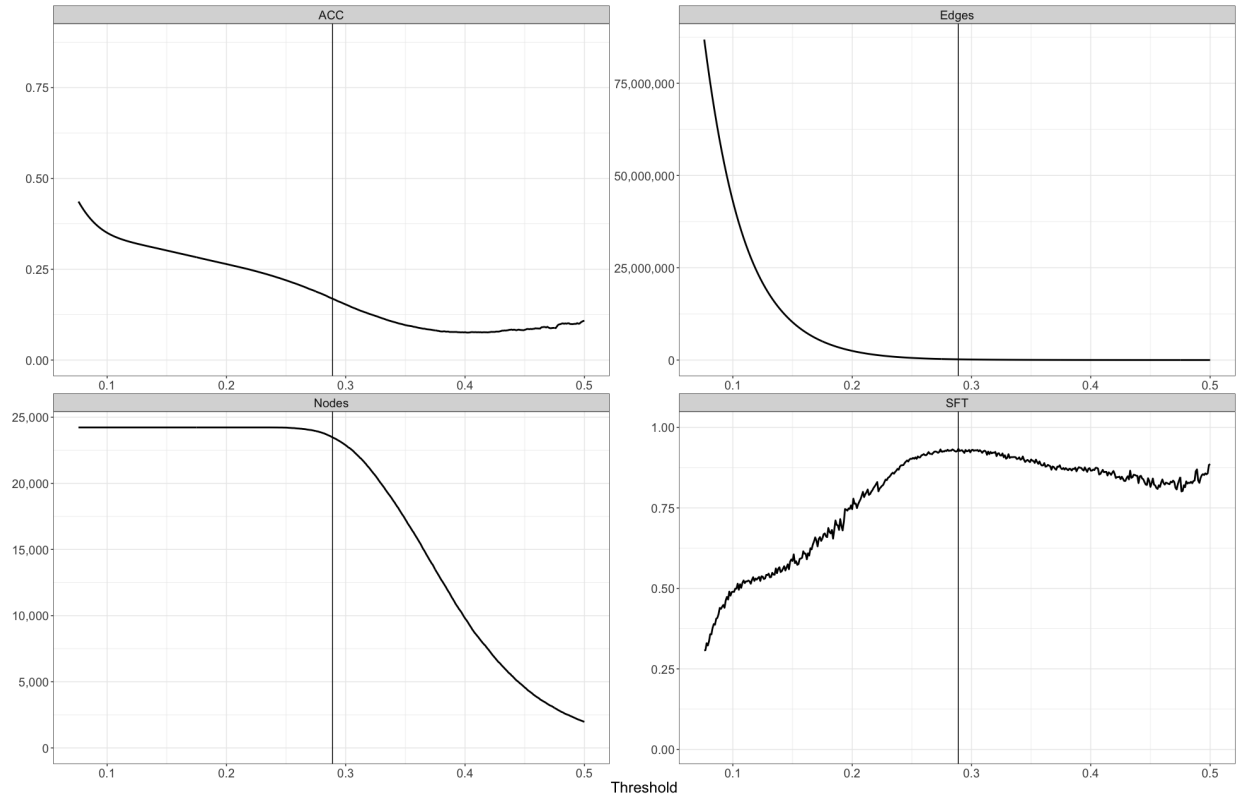
6.1 Running `seidr threshold`

The goal of `seidr threshold` is to provide a utility that assists in picking a hard cutoff. To that end it will iterate the network through a list of predefined thresholds and calculate:

- The number of edges
- The number of nodes
- The R^2 fit of the network to the Scale Free Distribution
- The Average Clustering Coefficient

It is left to the user to determine the final cutoff, based on expectation and background knowledge of the network.

In this example, we have already filtered the nodes of the network to strip away those of low interest, our goal is therefore to maximize the number of nodes kept, while keeping SFT and ACC high. At the indicated value, we keep 23470 nodes, 191547 edges, with a SFT of 0.933 and an ACC of 0.169:



6.2 A note on "scale freeness"

Using either scale freeness or average clustering coefficient to determine a hard cut for a network is not without issues. Recent insights (e.g. [Broido2018]) show that scale free networks are rare in real world networks and the criterion should most definitely be applied with caution. A better approach would be to select nodes kept from a known "gold standard" or - if only the core interactions are of interest - to perform "Network backboning" as described in e.g. [Coscia2017] .

6.3 Running `seidr threshold`

`seidr threshold` takes as a minimum a `SeidrFile` - usually, but not necessarily - from an aggregated network. By default it will create 1,000 evenly spaced thresholds in range $[0, 1]$. In practice, this tends to be wasteful of resources as most high density thresholds tend to be not useful. There are several options to adjust the range of thresholds to be tested:

- `--min, -m`: Adjust the lowest threshold to be tested
- `--max, -M`: Adjust the highest threshold to be tested
- `--nsteps, -n`: Adjust the number of steps to be tested

By default, `seidr threshold` will check the *score* of the last column in the `SeidrFile`, which can be adjusted with:

- `--index, -i`: Adjust which score to use to determine the cutoff
- `--threshold-rank, -r`: Determine a rank cutoff as opposed to a score

Be mindful that if you choose to threshold ranks, the meaning of minimum and maximum change (in the rank, lower is better). And to adjust the ranges not to test in rage $[0, 1]$, but rather $[1, N]$, where N is the highest number of edges you would like to test.

6.4 Output

`seidr threshold` writes a tab separated table to stdout. The headers are:

- Threshold
- Number of nodes
- Number of edges
- Scale free fit (R^2)
- Average clustering coefficient

An example output looks like:

```
0.842 5 5 0.80504672  0.375
0.841 5 5 0.80504672  0.375
0.84  5 5 0.80504672  0.375
0.839 5 6 0.027420548 0.54545455
0.838 5 6 0.027420548 0.54545455
0.837 5 6 0.027420548 0.54545455
0.836 5 6 0.027420548 0.54545455
0.835 7 7 0.93935184  0.54545455
0.834 7 7 0.93935184  0.54545455
```

CHAPTER 7

Calculating a Network Backbone

`seidr` implements Coscia and Neffke's backboning algorithm (very neatly described here if you don't feel like handling a lot of math, otherwise here: [Coscia2017]).

On any `SeidrFile` you can run:

```
seidr backbone seidrfile.sf
```

to calculate the network backbone statistics. Not that we are not cutting edges yet. To do that, we need to specify a measure of standard deviations to cut. Essentially, we want to define how extreme an edge has to deviate from its expected value, so that we keep it, the higher, the more stringent. A conservative value, would be 1.28, which corresponds approxiamtely to a P-value of 0.1:

```
seidr backbone -F 1.28 -o seidrfile.bb.1.28.sf seidrfile.bb.sf
```

CHAPTER 8

Getting network statistics

8.1 Centrality

8.1.1 How scores are used in `seidr stats`

We typically use scores as measures of similarity in `seidr` workflows. This means that *higher is better*. As an example in this network:

```
A  B  1
A  C  0.5
```

the edge A<->B is *stronger* than A<->C. In centrality metrics we often use weights as either similarity, or distance. When we e.g., calculate betweenness centrality, we want to know the shortest path from A to B, therefore weights are usually interpreted as distances here, and therefor *lower is better*.

By default `seidr` assumes that weights are *similarities* and handles them as such. When sensible, it will use $\frac{1}{w}$. If your data represents a distance, you must use the flag `--weight-is-distance`, otherwise your outcome will be wrong. If you set this flag, seidr will use $\frac{1}{w}$ for calculations where it assumes the weight indicates a *similarity* (i.e. the behaviour is inverse). See metrics below where the similarity [S] and distance [D] metrics are indicated.

8.1.2 Metrics

`seidr` can calculate a limited number of network centrality statistics on `SeidrFiles`.

On any `SeidrFile` you can run:

```
seidr stats seidrfile.sf
```

to calculate the network centrality statistics. By default all metrics that can be calculated will be. Use the `-m, --metrics` option to control this (see above as to the meaning of [S] and [D]):

For nodes

- `PR` - PageRank [S]

- `CLO` - Closeness centrality [D]
- `BTW` - Betweenness centrality [D]
- `STR` - Strength (weighted degree) centrality [S]
- `EV` - Eigenvector centrality [S]
- `KTZ` - Katz centrality [S]
- `LPC` - Laplacian centrality [S]

For edges

- `SEC` - Spanning edge centrality [D]
- `EBC` - Edge betweenness centrality [D]

To select only few of these run e.g.:

```
stats stats -m BTW,CLO seidrfile.sf
```

8.1.3 Approximate vs exact

By default, `seidr` uses approximations where it can to compute centrality statistics. It will sample `-n, --nsamples` nodes to do so. If not specified, that number is 10% of nodes. If your network is small, you can turn on exact metrics with `-e, --exact`.

8.1.4 Viewing stats

You can view node level statistics with:

```
seidr view --centrality seidrfile.sf
```

Edge level statistics are stored as edge attributes. You can add tags to see which attributes correspond to which stat:

```
seidr view -a seidrfile.sf
```

CHAPTER 9

ANOVERENCE

Please not that it is currently not recommended to run ANOVERENCE due to inconsitencies with the original implementation that we were not able to clarify with the original author

ANOVERENCE ([Kueffner2012]) employs the η^2 metric, a nonlinear correlation coefficient derived from an analysis of variance (ANOVA) ([Cohen1973]). It is one of the few methods that make direct use of experiment metadata, like perturbations, knockouts and overexpressions.

9.1 Running ANOVERENCE

ANOVERENCE needs a minimum of three input files:

- `-i, --infile`: An expression matrix (genes are columns, samples are rows) without headers.
- `-g, --genes`: A file containing gene names that correspond to columns in the expression matrix.
- `-e, --features`: A file that contains experiment metadata.

Here is an example matrix containing expression data for five genes in ten samples:

```
0.4254475 0.0178292 0.9079888 0.4482474 0.1723238
0.4424002 0.0505248 0.8693676 0.4458513 0.1733112
1.0568470 0.2084539 0.4674478 0.5050774 0.2448833
1.1172264 0.0030010 0.3176543 0.3872039 0.2537921
0.9710677 0.0010565 0.3546514 0.4745322 0.2077183
1.1393856 0.1220468 0.4024654 0.3484362 0.1686139
1.0648694 0.1405077 0.4817628 0.4748571 0.1826433
0.8761173 0.0738140 1.0582917 0.7303661 0.0536562
1.2059661 0.1534070 0.7608608 0.6558457 0.1577311
1.0006755 0.0789863 0.8036309 0.8389751 0.0883061
```

In the genes files, we provide the column headers for the expression matrix *in order*:

```
G1
G2
G3
G4
G5
```

The metadata file contains eight columns plus one row for each sample. If a column is not applicable, provide `NA` as input. Note that this file has headers:

```
Experiment Perturbations PerturbationLevels  Treatment DeletedGenes ␣
↪OverexpressedGenes  Time  Repeat
1 NA  NA  NA  NA  NA  NA  1
1 NA  NA  NA  NA  NA  NA  2
2 NA  NA  NA  NA  NA  NA  1
3 NA  NA  NA  NA  NA  NA  1
3 NA  NA  NA  NA  NA  NA  2
4 NA  NA  NA  NA  NA  NA  1
4 NA  NA  NA  NA  NA  NA  2
5 NA  NA  NA  NA  NA  NA  1
5 NA  NA  NA  NA  NA  NA  2
5 NA  NA  NA  NA  NA  NA  3
```

Further we need to provide a `-w, --weight`, typically an integral value between 10 and 1000 that controls how much more weight we give to perturbation experiments that involve the genes that are tested. Once we have those four parameters, we are ready to run `anoverence`:

```
anoverence -i exr_mat.tsv -g genes.txt -e meta.tsv -w 500
```

As output we receive a lower triangular matrix of interaction scores:

```
0.288087
0.388856        0.405731
0.459865        0.276648        0.336653
0.432748        0.374432        0.397973        0.403535
```

9.2 Running ANOVERENCE for a subset of genes

Often we have only a small number of genes of interest. We can instruct `ANOVERENCE` to only calculate interactions involving those genes by providing a `-t, --targets` file containing these gene names:

```
G3
G4
```

And running it with the `-t, --targets` options:

```
anoverence -i expr_mat.tsv -g genes.txt -e meta.tsv -w 500 -t targets.txt
```

In this case we will receive an edge list as output:

```
G3  G1  0.388856
G4  G1  0.459865
G3  G2  0.405731
G4  G2  0.276648
G4  G3  0.336653
```

(continues on next page)

(continued from previous page)

```
G3  G5  0.397973
G4  G5  0.403535
```

CHAPTER 10

ARACNE

ARACNE ([Margolin2006]) is an inference algorithm based on mutual information and applies data processing inequality to delete most indirect edges.

Our implementation differs to the original in that it estimates the initial mutual information using a B-spline approach as described in [Daub2004] .

10.1 Running ARACNE

ARACNE needs a minimum of two input files:

- `-i, --infile`: An expression matrix (genes are columns, samples are rows) without headers.
- `-g, --genes`: A file containing gene names that correspond to columns in the expression matrix.

Here is an example matrix containing expression data for five genes in ten samples:

```
0.4254475 0.0178292 0.9079888 0.4482474 0.1723238
0.4424002 0.0505248 0.8693676 0.4458513 0.1733112
1.0568470 0.2084539 0.4674478 0.5050774 0.2448833
1.1172264 0.0030010 0.3176543 0.3872039 0.2537921
0.9710677 0.0010565 0.3546514 0.4745322 0.2077183
1.1393856 0.1220468 0.4024654 0.3484362 0.1686139
1.0648694 0.1405077 0.4817628 0.4748571 0.1826433
0.8761173 0.0738140 1.0582917 0.7303661 0.0536562
1.2059661 0.1534070 0.7608608 0.6558457 0.1577311
1.0006755 0.0789863 0.8036309 0.8389751 0.0883061
```

In the genes files, we provide the column headers for the expression matrix *in order*:

```
G1
G2
G3
G4
G5
```

With that, we can run ARACNE:

```
mi -m ARACNE -i expr_mat.tsv -g genes.txt
```

The output is a lower triangular matrix of scores:

```
0
0.798215    0.874873
0           0.889133    0
0           0           0.860645    0.95965
```

10.2 Tuning the number of bins and spline degree

Estimating mutual infofrmation from discrete data is well defined, but normalized expression data is usually continuous. To estimate the MI from continuous data, each data point is usually assigned to one bin. This can lead to a loss of information.

The B-Spline estimator for MI therefore performs fuzzy assignment of the data to bins. The `-s, --spline` parameter controls the spline degree (therefore the shape) of the indicator function. For `s=1` the indicator function is the same as for simple binning. Improvements in the MI beyond a degree of `s=3` are rarely seen, therefore it is a good choice as a default.

The number of bins used in the assignment can be controlled with the `-b, --bins` option. By default it is automatically inferred from the data, but this can lead to high memory requirements on large datasets. Generally, the number of bins is assumed not to influence the MI much as long as it's within a reasonable range. A value between 5 and 10 is a good starting point for typically sized datasets from RNA-Seq.

10.3 Running ARACNE for a subset of genes

Often we have only a small number of genes of interest. We can instruct ARACNE to only calculate interactions involving those genes by providing a `-t, --targets` file containing these gene names:

```
G3
G4
```

And running it with the `-t, --targets` options:

```
mi -m ARACNE -i expr_mat.tsv -g genes.txt -t targets.txt
```

In this case we will receive an edge list as output:

```
G3  G1  0.798215
G3  G2  0.874873
G3  G4  0
G3  G5  0.860645
G4  G1  0
G4  G2  0.889133
G4  G3  0
G4  G5  0.95965
```

10.4 Running ARACNE in MPI mode

ARACNE can use parallel processing in the MI estimation step. For general info on how to run parallel algorithms in `seidr`, please see *Using multiple processors to infer networks*

CHAPTER 11

CLR

CLR ([Faith2007]) is an inference algorithm based on mutual information and applies contextual likelihood of relatedness to reweight edges based on a shared neighbourhood.

Our implementation estimates the initial mutual information using a B-spline approach as described in [Daub2004] .

11.1 Running CLR

CLR needs a minimum of two input files:

- `-i, --infile`: An expression matrix (genes are columns, samples are rows) without headers.
- `-g, --genes`: A file containing gene names that correspond to columns in the expression matrix.

Here is an example matrix containing expression data for five genes in ten samples:

```
0.4254475 0.0178292 0.9079888 0.4482474 0.1723238
0.4424002 0.0505248 0.8693676 0.4458513 0.1733112
1.0568470 0.2084539 0.4674478 0.5050774 0.2448833
1.1172264 0.0030010 0.3176543 0.3872039 0.2537921
0.9710677 0.0010565 0.3546514 0.4745322 0.2077183
1.1393856 0.1220468 0.4024654 0.3484362 0.1686139
1.0648694 0.1405077 0.4817628 0.4748571 0.1826433
0.8761173 0.0738140 1.0582917 0.7303661 0.0536562
1.2059661 0.1534070 0.7608608 0.6558457 0.1577311
1.0006755 0.0789863 0.8036309 0.8389751 0.0883061
```

In the genes files, we provide the column headers for the expression matrix *in order*:

```
G1
G2
G3
G4
G5
```

With that, we can run CLR:

```
mi -m CLR -i expr_mat.tsv -g genes.txt
```

The output is a lower triangular matrix of scores:

```
0.320993
0.944725    0.858458
0.431752    0.9078      0.453098
0.0897561   0.579328    0.794528    1.15506
```

11.2 Tuning the number of bins and spline degree

Estimating mutual infofrmation from discrete data is well defined, but normalized expression data is usually continuous. To estimate the MI from continuous data, each data point is usually assigned to one bin. This can lead to a loss of information.

The B-Spline estimator for MI therefore performs fuzzy assignment of the data to bins. The `-s, --spline` parameter controls the spline degree (therefore the shape) of the indicator function. For `s=1` the indicator function is the same as for simple binning. Improvements in the MI beyond a degree of `s=3` are rarely seen, therefore it is a good choice as a default.

The number of bins used in the assignment can be controlled with the `-b, --bins` option. By default it is automatically inferred from the data, but this can lead to high memory requirements on large datasets. Generally, the number of bins is assumed not to influence the MI much as long as it's within a reasonable range. A value between 5 and 10 is a good starting point for typically sized datasets from RNA-Seq.

11.3 Running CLR for a subset of genes

Often we have only a small number of genes of interest. We can instruct CLR to only calculate interactions involving those genes by providing a `-t, --targets` file containing these gene names:

```
G3
G4
```

And running it with the `-t, --targets` options:

```
mi -m CLR -i expr_mat.tsv -g genes.txt -t targets.txt
```

In this case we will receive an edge list as output:

```
G3  G1  0.944725
G3  G2  0.858458
G3  G4  0.453098
G3  G5  0.794528
G4  G1  0.431752
G4  G2  0.9078
G4  G3  0.453098
G4  G5  1.15506
```

11.4 Running CLR in MPI mode

CLR can use parallel processing in the MI estimation step. For general info on how to run parallel algorithms in `seidr`, please see *Using multiple processors to infer networks*

CHAPTER 12

Correlation

This simple executable calculates pearson or spearman correlation from a set of expression data.

12.1 Running correlation

Correlation needs a minimum of two input files:

- `-i, --infile`: An expression matrix (genes are columns, samples are rows) without headers.
- `-g, --genes`: A file containing gene names that correspond to columns in the expression matrix.

Here is an example matrix containing expression data for five genes in ten samples:

```
0.4254475 0.0178292 0.9079888 0.4482474 0.1723238
0.4424002 0.0505248 0.8693676 0.4458513 0.1733112
1.0568470 0.2084539 0.4674478 0.5050774 0.2448833
1.1172264 0.0030010 0.3176543 0.3872039 0.2537921
0.9710677 0.0010565 0.3546514 0.4745322 0.2077183
1.1393856 0.1220468 0.4024654 0.3484362 0.1686139
1.0648694 0.1405077 0.4817628 0.4748571 0.1826433
0.8761173 0.0738140 1.0582917 0.7303661 0.0536562
1.2059661 0.1534070 0.7608608 0.6558457 0.1577311
1.0006755 0.0789863 0.8036309 0.8389751 0.0883061
```

In the genes files, we provide the column headers for the expression matrix *in order*:

```
G1
G2
G3
G4
G5
```

With that, we can run correaltion in Pearson mode:

```
correlation -m pearson -i expr_mat.tsv -g genes.txt
```

or in Spearman mode:

```
correlation -m spearman -i expr_mat.tsv -g genes.txt
```

The output is a lower triangular matrix of scores:

```
0.469355
-0.587163   -0.0704821
0.127765    0.16474     0.597376
0.145338    0.0138744   -0.77125    -0.758263
```

12.2 Optional arguments to correlation

- `-a, --absolute`: By default, the executable reports signed correlation values. Using this option will turn on reporting of the absolute value of the correlation coefficient. It is generally recommended to export correlation with signs (i.e. *not* absolute) and instead run `seidr import` in absolute mode, which will rank genes by their magnitude, but won't throw away the sign information.
- `-s, --scale`: This triggers feature scaling of the expression matrix before the correlation calculation. Generally this should be *on* especially when calculating Pearson's rho.

12.3 Running Correlation for a subset of genes

Often we have only a small number of genes of interest. We can instruct correlation to only calculate interactions involving those genes by providing a `-t, --targets` file containing these gene names:

```
G3
G4
```

And running it with the `-t, --targets` options:

```
correlation -i expr_mat.tsv -g genes.txt -t targets.txt
```

In this case we will receive an edge list as output:

```
G3  G1  -0.587163
G3  G2  -0.0704821
G3  G4  0.597376
G3  G5  -0.77125
G4  G1  0.127765
G4  G2  0.16474
G4  G3  0.597376
G4  G5  -0.758263
```

CHAPTER 13

Ensemble methods

The ensemble methods are based on [Ruyssinck2014] . The three main executables work the same way and have the same options. They all work by resampling the expression data along samples and genes, which often reduces variance in their predictions:

- `el-ensemble`: Uses an ensemble of Elastic Net regression predictors.
- `svm-ensemble`: Uses an ensemble of Support Vector Machine predictors.
- `llr-ensemble`: Uses an ensemble of Support Vector Machine predictors.

The Elastic Net code uses the GLMNET Fortran backend from [Friedman2010] .

13.1 Running Ensembles

Each ensemble needs a minimum of two input files:

- `-i, --infile`: An expression matrix (genes are columns, samples are rows) without headers.
- `-g, --genes`: A file containing gene names that correspond to columns in the expression matrix.

Here is an example matrix containing expression data for five genes in ten samples:

```
0.4254475 0.0178292 0.9079888 0.4482474 0.1723238
0.4424002 0.0505248 0.8693676 0.4458513 0.1733112
1.0568470 0.2084539 0.4674478 0.5050774 0.2448833
1.1172264 0.0030010 0.3176543 0.3872039 0.2537921
0.9710677 0.0010565 0.3546514 0.4745322 0.2077183
1.1393856 0.1220468 0.4024654 0.3484362 0.1686139
1.0648694 0.1405077 0.4817628 0.4748571 0.1826433
0.8761173 0.0738140 1.0582917 0.7303661 0.0536562
1.2059661 0.1534070 0.7608608 0.6558457 0.1577311
1.0006755 0.0789863 0.8036309 0.8389751 0.0883061
```

In the genes files, we provide the column headers for the expression matrix *in order*:

```
G1
G2
G3
G4
G5
```

With that, we can run the ensembles:

```
el-ensemble -i expr_mat.tsv -g genes.txt
svm-ensemble -i expr_mat.tsv -g genes.txt
llr-ensemble -i expr_mat.tsv -g genes.txt
```

The output is a square matrix of scores:

```
0       0   0.876   0.124   0
0.894   0   0.106   0       0
0.894   0   0       0.106   0
0.894   0   0.106   0       0
0.894   0   0.106   0       0
```

13.2 Optional arguments for the Ensemble methods

- `-s, --scale`: This triggers feature scaling of the expression matrix before the regression calculation. Generally this should be *on*.
- `-X, --max-experiment-size`: In each resampling iteration, choose maximally this many samples along rows (experiments) of the dataset.
- `-x, --min-experiment-size`: In each resampling iteration, choose minimally this many samples along rows (experiments) of the dataset.
- `-P, --max-predictor-size`: In each resampling iteration, choose maximally this many genes along columns (predictors) of the dataset.
- `-p, --min-predictor-size`: In each resampling iteration, choose minimally this many genes along columns (predictors) of the dataset.
- `-e, --ensemble`: Perform this many resampling iterations for each gene.

The sampling boundaries `-X`, `-x`, `-P` and `-p` default to 4/5th of samples/predictors for the upper bound and 1/5th for the lower. In runs with small experiment sizes (<50) one should set this manually higher to avoid undersampling. In these cases, I suggest 90% for the upper boundary and 50% for the lower (in experiments). These are *absolute* numbers. E.g., if you have 50 samples and you want 50% - 90% as a lower & upper bound, set `-x 25 -X 45`.

13.3 Running ensembles for a subset of genes

Often we have only a small number of genes of interest. We can instruct the ensembles to only calculate interactions involving those genes by providing a `-t, --targets` file containing these gene names:

```
G3
G4
```

And running it with the `-t, --targets` options:

```
llr-ensemble -i expr_mat.tsv -g genes.txt -t targets.txt
svm-ensemble -i expr_mat.tsv -g genes.txt -t targets.txt
el-ensemble -i expr_mat.tsv -g genes.txt -t targets.txt
```

In this case we will receive an edge list as output:

```
G3  G1  0.894
G3  G2  0
G3  G4  0.106
G3  G5  0
G4  G1  0.894
G4  G2  0
G4  G3  0.106
G4  G5  0
```

13.4 Running Ensembles in MPI mode

Each ensemble can use parallel processing. For general info on how to run parallel algorithms in `seidr`, please see *Using multiple processors to infer networks*

13.5 The difference between SVM and LLR

LLR and SVM are based on different implementations of SVMs in C. One is based on LibLinear , the other on LibSVM using a linear kernel. While they should in general agree most of the time, coefficients are handled differently. SVM is closer to the reference implementation by [Ruyssinck2014] , but LLR is much faster.

CHAPTER 14

GENIE3

GENIE3 calculates a Random Forest regression using genes as predictors. It then uses Random Forests importance measures as gene association scores. The method is described in [Huynh-Thu2010] . Internally, our implementation uses ranger to calculate the forests and importance [Wright2017] .

14.1 Running GENIE3

GENIE3 needs a minimum of two input files:

- `-i, --infile`: An expression matrix (genes are columns, samples are rows) without headers.
- `-g, --genes`: A file containing gene names that correspond to columns in the expression matrix.

Here is an example matrix containing expression data for five genes in ten samples:

```
0.4254475 0.0178292 0.9079888 0.4482474 0.1723238
0.4424002 0.0505248 0.8693676 0.4458513 0.1733112
1.0568470 0.2084539 0.4674478 0.5050774 0.2448833
1.1172264 0.0030010 0.3176543 0.3872039 0.2537921
0.9710677 0.0010565 0.3546514 0.4745322 0.2077183
1.1393856 0.1220468 0.4024654 0.3484362 0.1686139
1.0648694 0.1405077 0.4817628 0.4748571 0.1826433
0.8761173 0.0738140 1.0582917 0.7303661 0.0536562
1.2059661 0.1534070 0.7608608 0.6558457 0.1577311
1.0006755 0.0789863 0.8036309 0.8389751 0.0883061
```

In the genes files, we provide the column headers for the expression matrix *in order*:

```
G1
G2
G3
G4
G5
```

With that, we can run GENIE3:

```
genie3 -i expr_mat.tsv -g genes.txt
```

The output is a square matrix of scores:

```
0           0.108322    0.264794    0.0692147   0.0482803
0.00914761  0           0.00844504  0.00974063  0.00616896
0.167265    0.0721168   0           0.0914891   0.180078
0.0163077   0.0211425   0.0369387   0           0.0932622
0.00277062  0.00334468  0.00934115  0.0114427   0
```

14.2 Optional arguments for GENIE3

- `-s, --scale`: This triggers feature scaling of the expression matrix before the regression calculation. Generally this should be *off*.
- `-b, --ntree`: Grow this many trees for each gene.
- `-m, --mtry`: Sample this many features (genes) for each tree.
- `-p, --min-prop`: Lower quantile of covariate distribution to be considered for splitting.
- `-a, --alpha`: Significance threshold to allow splitting.
- `-N, --min-node-size`: Minimum node size

14.3 Running GENIE3 for a subset of genes

Often we have only a small number of genes of interest. We can instruct GENIE3 to only calculate interactions involving those genes by providing a `-t, --targets` file containing these gene names:

```
G3
G4
```

And running it with the `-t, --targets` options:

```
genie3 -i expr_mat.tsv -g genes.txt -t targets.txt
```

In this case we will receive an edge list as output:

```
G3  G1  0.167265
G3  G2  0.0721168
G3  G4  0.0914891
G3  G5  0.180078
G4  G1  0.0163077
G4  G2  0.0211425
G4  G3  0.0369387
G4  G5  0.0932622
```

14.4 Running GENIE3 in MPI mode

GENIE3 can use parallel processing. For general info on how to run parallel algorithms in `seidr`, please see *Using multiple processors to infer networks*

CHAPTER 15

NARROMI

Narromi is an MI based algorithm that tries to minimize noise in the MI using linear programming. It is published in [Zhang2013] .

15.1 Running Narromi

Narromi needs a minimum of two input files:

- `-i, --infile`: An expression matrix (genes are columns, samples are rows) without headers.
- `-g, --genes`: A file containing gene names that correspond to columns in the expression matrix.

Here is an example matrix containing expression data for five genes in ten samples:

```
0.4254475 0.0178292 0.9079888 0.4482474 0.1723238
0.4424002 0.0505248 0.8693676 0.4458513 0.1733112
1.0568470 0.2084539 0.4674478 0.5050774 0.2448833
1.1172264 0.0030010 0.3176543 0.3872039 0.2537921
0.9710677 0.0010565 0.3546514 0.4745322 0.2077183
1.1393856 0.1220468 0.4024654 0.3484362 0.1686139
1.0648694 0.1405077 0.4817628 0.4748571 0.1826433
0.8761173 0.0738140 1.0582917 0.7303661 0.0536562
1.2059661 0.1534070 0.7608608 0.6558457 0.1577311
1.0006755 0.0789863 0.8036309 0.8389751 0.0883061
```

In the genes files, we provide the column headers for the expression matrix *in order*:

```
G1
G2
G3
G4
G5
```

With that, we can run Narromi:

```
narromi -i expr_mat.tsv -g genes.txt
```

The output is a square matrix of scores:

```
0           0           0.581267    0.00822935  0.0106747
0.100319    0           0.00249005  0.0137571   9.62593e-05
0.116941    0.00249005  0           0.624368    0
0.00822935  0.0137571   0.50236     0           0
0.0106747   9.62593e-05 0.29456     0.199657    0
```

15.2 Optional arguments for Narromi

- `-a, --alpha`: Initial cutoff for MI selection (alpha).
- `-m, --algorithm`: Linear programming algorithm. Interior point is probably faster, but can be unstable for some datasets. When in doubt, choose simplex.

15.3 Running Narromi for a subset of genes

Often we have only a small number of genes of interest. We can instruct Narromi to only calculate interactions involving those genes by providing a `-t, --targets` file containing these gene names:

```
G3
G4
```

And running it with the `-t, --targets` options:

```
narromi -i expr_mat.tsv -g genes.txt -t targets.txt
```

In this case we will receive an edge list as output:

```
G3  G1  0.116941
G3  G2  0.00249005
G3  G4  0.624368
G3  G5  0
G4  G1  0.00822935
G4  G2  0.0137571
G4  G3  0.50236
G4  G5  0
```

15.4 Running Narromi in MPI mode

Narromi can use parallel processing. For general info on how to run parallel algorithms in `seidr`, please see *Using multiple processors to infer networks*

CHAPTER 16

Partial Correlation

PCor is an MI based algorithm that tries to minimize noise in the MI using linear programming. It is published in [Schafer2005] .

16.1 Running PCor

PCor needs a minimum of two input files:

- `-i, --infile`: An expression matrix (genes are columns, samples are rows) without headers.
- `-g, --genes`: A file containing gene names that correspond to columns in the expression matrix.

Here is an example matrix containing expression data for five genes in ten samples:

```
0.4254475 0.0178292 0.9079888 0.4482474 0.1723238
0.4424002 0.0505248 0.8693676 0.4458513 0.1733112
1.0568470 0.2084539 0.4674478 0.5050774 0.2448833
1.1172264 0.0030010 0.3176543 0.3872039 0.2537921
0.9710677 0.0010565 0.3546514 0.4745322 0.2077183
1.1393856 0.1220468 0.4024654 0.3484362 0.1686139
1.0648694 0.1405077 0.4817628 0.4748571 0.1826433
0.8761173 0.0738140 1.0582917 0.7303661 0.0536562
1.2059661 0.1534070 0.7608608 0.6558457 0.1577311
1.0006755 0.0789863 0.8036309 0.8389751 0.0883061
```

In the genes files, we provide the column headers for the expression matrix *in order*:

```
G1
G2
G3
G4
G5
```

With that, we can run PCor:

```
pcor -i expr_mat.tsv -g genes.txt
```

The output is a lower triangular matrix of scores:

```
0.291919
-0.431942   0.0617938
0.218244    0.0683963   0.266362
-0.0361338  0.0472015   -0.363056   -0.361116
```

16.2 Optional arguments for PCor

- `-a, --absolute`: By default, the executable reports signed correlation values. Using this option will turn on reporting of the absolute value of the correlation coefficient. It is generally recommended to export correlation with signs (i.e. *not* absolute) and instead run `seidr import` in absolute mode, which will rank genes by their magnitude, but won't throw away the sign information.

16.3 Running PCor for a subset of genes

Often we have only a small number of genes of interest. We can instruct PCor to only calculate interactions involving those genes by providing a `-t, --targets` file containing these gene names:

```
G3
G4
```

And running it with the `-t, --targets` options:

```
pcor -i expr_mat.tsv -g genes.txt -t targets.txt
```

In this case we will receive an edge list as output:

```
G3  G1  -0.431942
G3  G2  0.0617938
G3  G4  0.266362
G3  G5  -0.363056
G4  G1  0.218244
G4  G2  0.0683963
G4  G3  0.266362
G4  G5  -0.361116
```

CHAPTER 17

PLSNET

PLSNET uses a partial least squares feature selection algorithm to predict interacting genes. It is published in [Guo2016] .

17.1 Running PLSNET

PLSNET needs a minimum of two input files:

- `-i, --infile`: An expression matrix (genes are columns, samples are rows) without headers.
- `-g, --genes`: A file containing gene names that correspond to columns in the expression matrix.

Here is an example matrix containing expression data for five genes in ten samples:

```
0.4254475 0.0178292 0.9079888 0.4482474 0.1723238
0.4424002 0.0505248 0.8693676 0.4458513 0.1733112
1.0568470 0.2084539 0.4674478 0.5050774 0.2448833
1.1172264 0.0030010 0.3176543 0.3872039 0.2537921
0.9710677 0.0010565 0.3546514 0.4745322 0.2077183
1.1393856 0.1220468 0.4024654 0.3484362 0.1686139
1.0648694 0.1405077 0.4817628 0.4748571 0.1826433
0.8761173 0.0738140 1.0582917 0.7303661 0.0536562
1.2059661 0.1534070 0.7608608 0.6558457 0.1577311
1.0006755 0.0789863 0.8036309 0.8389751 0.0883061
```

In the genes files, we provide the column headers for the expression matrix *in order*:

```
G1
G2
G3
G4
G5
```

With that, we can run PCor:

```
plsnet -i expr_mat.tsv -g genes.txt
```

The output is a square matrix of scores:

```
0       10661.7 9103.01 3781.48 8553.33
1672.03 0       3808.75 4130.81 24318.7
2783.44 6850.63 0       2885.31 23882.1
1683.27 11640.3 4560.34 0       63590
1218.51 20635.5 9127.68 14218.4 0
```

17.2 Optional arguments for PLSNET

- `-s, --scale`: This triggers feature scaling of the expression matrix before the correlation calculation. Generally this should be *on*.
- `-e, --ensemble`: Perform this many resampling iterations for each gene.
- `-c, --components`: The number of PLS components to be considered.
- `-p, --predictor-size`: The number of predictors (genes) to be sampled at each iteration.

17.3 Running PLSNET for a subset of genes

Often we have only a small number of genes of interest. We can instruct PLSNET to only calculate interactions involving those genes by providing a `-t, --targets` file containing these gene names:

```
G3
G4
```

And running it with the `-t, --targets` options:

```
plsnet -i expr_mat.tsv -g genes.txt -t targets.txt
```

In this case we will receive an edge list as output:

```
G3  G1  1560.18
G3  G2  892.019
G3  G4  1471.69
G3  G5  203666
G4  G1  943.506
G4  G2  1515.68
G4  G3  5611.66
G4  G5  542294
```

17.4 Running PLSNET in MPI mode

PLSNET can use parallel processing. For general info on how to run parallel algorithms in `seidr`, please see *Using multiple processors to infer networks*

CHAPTER 18

TIGRESS

TIGRESS uses an ensemble approach (here called stability selection) to reduce prediction variance in a LASSO model. It works somewhat similar to the other *Ensemble methods* . TIGRESS is published in [Haury2012] . The LASSO uses the GLMNET Fortran backend in [Friedman2010] .

18.1 Running TIGRESS

TIGRESS needs a minimum of two input files:

- `-i, --infile`: An expression matrix (genes are columns, samples are rows) without headers.
- `-g, --genes`: A file containing gene names that correspond to columns in the expression matrix.

Here is an example matrix containing expression data for five genes in ten samples:

```
0.4254475 0.0178292 0.9079888 0.4482474 0.1723238
0.4424002 0.0505248 0.8693676 0.4458513 0.1733112
1.0568470 0.2084539 0.4674478 0.5050774 0.2448833
1.1172264 0.0030010 0.3176543 0.3872039 0.2537921
0.9710677 0.0010565 0.3546514 0.4745322 0.2077183
1.1393856 0.1220468 0.4024654 0.3484362 0.1686139
1.0648694 0.1405077 0.4817628 0.4748571 0.1826433
0.8761173 0.0738140 1.0582917 0.7303661 0.0536562
1.2059661 0.1534070 0.7608608 0.6558457 0.1577311
1.0006755 0.0789863 0.8036309 0.8389751 0.0883061
```

In the genes files, we provide the column headers for the expression matrix *in order*:

```
G1
G2
G3
G4
G5
```

With that, we can run PCor:

```
tigress -i expr_mat.tsv -g genes.txt
```

The output is a square matrix of scores:

```
0       0.5197  0.6558  0.2139  0.0838
0.6909  0       0.169   0.2216  0.3628
0.4819  0.1774  0       0.3084  0.7075
0.137   0.1418  0.3349  0       0.675
0.0182  0.1736  0.7209  0.6138  0
```

18.2 Optional arguments for TIGRESS

- `-s, --scale`: This triggers feature scaling of the expression matrix before the correlation calculation. Generally this should be *on*.
- `-B, --nbootstrap`: Perform this many resampling iterations for each gene.
- `-n, --nlambda`: Consider this many shrinkage lambdas.
- `-l, --min-lambda`: The minimum lambda value considered is this fraction of the maximum.

18.3 Running TIGRESS for a subset of genes

Often we have only a small number of genes of interest. We can instruct TIGRESS to only calculate interactions involving those genes by providing a `-t, --targets` file containing these gene names:

```
G3
G4
```

And running it with the `-t, --targets` options:

```
tigress -i expr_mat.tsv -g genes.txt -t targets.txt
```

In this case we will receive an edge list as output:

```
G3  G1  0.4819
G3  G2  0.1774
G3  G4  0.3084
G3  G5  0.7075
G4  G1  0.137
G4  G2  0.1418
G4  G3  0.3349
G4  G5  0.675
```

18.4 Running TIGRESS in MPI mode

TIGRESS can use parallel processing. For general info on how to run parallel algorithms in `seidr`, please see *Using multiple processors to infer networks*

CHAPTER 19

TOM Similarity

The toplogical overlap matrix (TOM) is the similarity measure implemented by WGCNA [Langfelder2008]. It calculates a correlation matrix from the expression data, calculates a soft threshold and assigns two genes a high topological overlap if they share common neighbourhoods.

19.1 Running TOM similarity

`tomsimilarity` needs a minimum of two input files:

- `-i, --infile`: An expression matrix (genes are columns, samples are rows) without headers.
- `-g, --genes`: A file containing gene names that correspond to columns in the expression matrix.

Here is an example matrix containing expression data for five genes in ten samples:

```
6.107967  7.188796  7.139945  9.417835   6.195927
8.602925  9.134458  8.630118  10.695973  6.930023
6.699199  8.307864  8.174942  10.874148  7.143233
7.661777  8.891523  8.348661  10.439793  6.868748
7.031853  9.019152  8.539557  10.726523  7.461354
8.931517  9.246769  8.944240  10.774747  6.729316
6.815357  9.209684  8.607074  9.574451   7.400409
7.424712  9.603071  8.347164  10.609222  7.168921
8.465108  8.788967  8.875855  10.537852  6.628380
8.559188  8.992996  8.279209  10.640245  6.744078
```

In the genes files, we provide the column headers for the expression matrix *in order*:

```
G1
G2
G3
G4
G5
```

With that, we can run PCor:

```
tomsimilarity -i expr_mat.tsv -g genes.txt -b 4
```

The output is a lower triangular matrix of scores:

```
0.44357
0.486974  0.504881
0.370446  0.408224  0.42039
0.225011  0.465292  0.396999  0.252425
```

19.2 Optional arguments for `tomsimilarity`

- `-s, --scale`: This triggers feature scaling of the expression matrix before the correlation calculation. Generally this should be *on*.
- `-m, --method`: Choose between "pearson" or "bicor" (biweight midcorrelation. The latter is typically a good choice unless you have a lot of outliers.)
- `-b, --sft`: The soft threshold power. This is the exponent for soft thresholding the correlation matrix. Unless you know why, leave it default.
- `-M, --max-power`: When auto-detecting the soft threshold power, this is the maximum value that will be tested. It's usually not a good idea to go above 30. If you cannot get a good fit, decrease the cutoff instead.
- `-S, --sft-cutoff`: When the network reaches this scale free fit R^2 value, stop testing powers. Sometimes, you cannot get a good fit (>0.8) on larger datasets. In this case, decrease this value.
- `-T, --tom-type`: "unsigned", "signed", or "signed-hybrid". This defines how to score the TOM. "unsigned" is $|a_{ij}|$, "signed" is $\frac{a_{ij}+1}{2}$ and "signed-hybrid" is $|a_{ij}|$ for positive correlation, 0 otherwise.

19.3 Running `tomsimilarity` for a subset of genes

Often we have only a small number of genes of interest. We can instruct `tomsimilarity` to only calculate interactions involving those genes by providing a `-t, --targets` file containing these gene names:

```
G3
G4
```

And running it with the `-t, --targets` options:

```
tomsimilarity -i expr_mat.tsv -g genes.txt -t targets.txt -b 4
```

In this case we will receive an edge list as output:

```
G3  G1  0.486974
G3  G2  0.504881
G3  G4  0.42039
G3  G5  0.396999
G4  G1  0.370446
G4  G2  0.408224
G4  G3  0.42039
G4  G5  0.252425
```

CHAPTER 20

Comparing networks of different species with `seidr`

20.1 Introduction

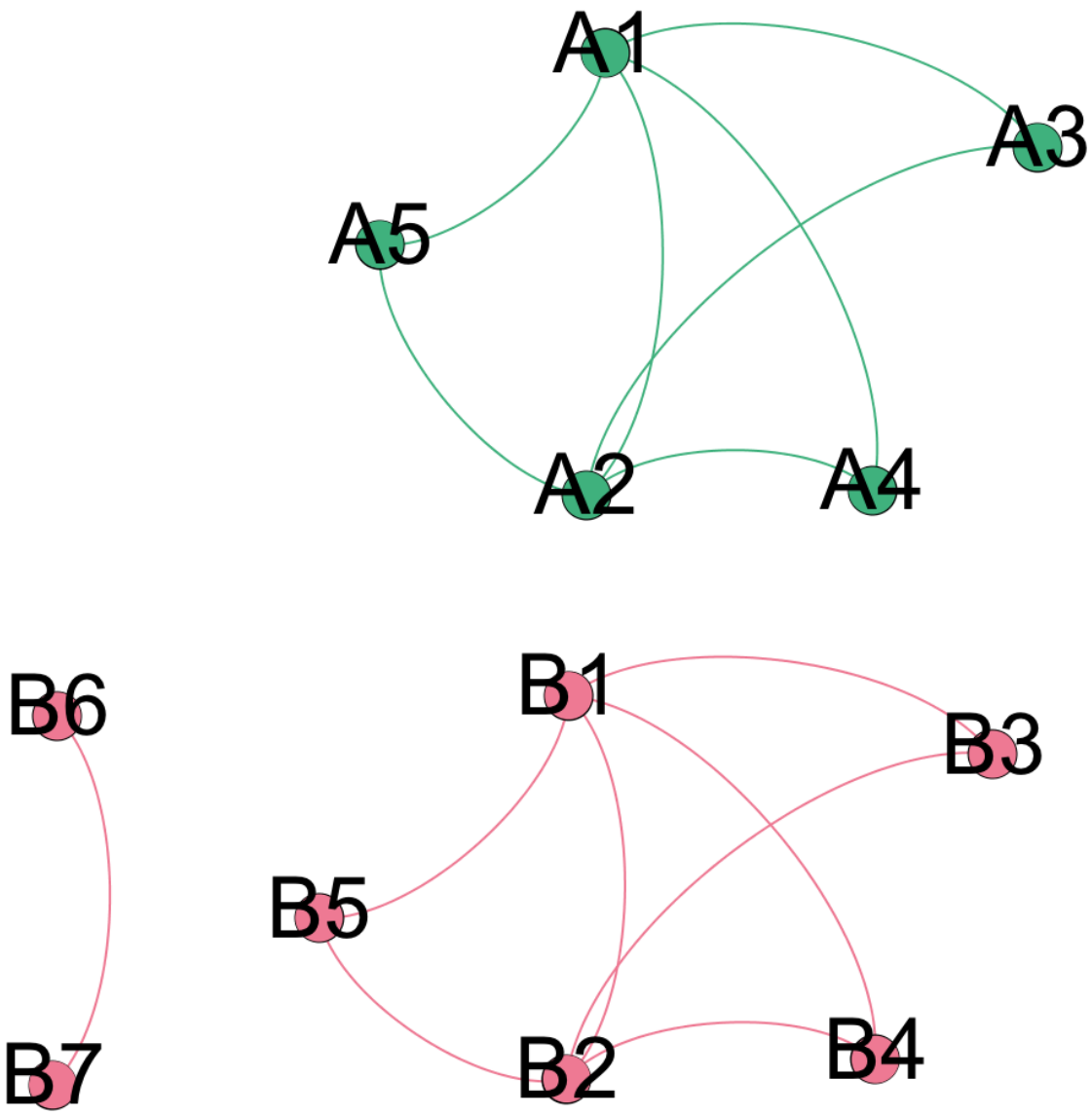
Seidr can compare edges and nodes of two networks that originate from separate species **if the user supplies an onthology** to translate the node IDs from network A to network B. Consider the these two networks:

net1.sf:

Source	Target	Type	Weight;Rank
A1	A2	Directed	7.82637e-06;7
A1	A3	Directed	0.131538;5
A2	A3	Directed	0.532767;2
A1	A4	Directed	0.755605;1
A2	A4	Directed	0.218959;4
A1	A5	Directed	0.45865;3
A2	A5	Directed	0.0470446;6

net2.sf:

Source	Target	Type	Weight;Rank
B1	B2	Directed	7.82637e-06;8
B1	B3	Directed	0.131538;6
B2	B3	Directed	0.532767;3
B1	B4	Directed	0.755605;1
B2	B4	Directed	0.218959;5
B1	B5	Directed	0.45865;4
B2	B5	Directed	0.0470446;7
B6	B7	Directed	0.678865;2



Before we can overlap these two networks, we need to define which nodes are equivalent between them. The file format is a very simple TAB delimited dictionary, each line defining a translation from A to B:

```
A1  B1
A2  B2
A3  B3
A4  B4
A5  B5
```

20.2 Important info

- The compare function currently completely ignores directionality. All output will be undirected.

- There is no support for asymmetric translations. If A1 -> B1, but B1 -> A2 it is left to the user which translation to prioritize.
- Ranks will be merged via $\sum A_{ij}B_{ij}$ for overlapping edges where A_{ij} is an edge in network A and B_{ij} is an edge in network B
- Scores will be computed from all ranks in the dataset via $\frac{x_i - min(x)}{max(x) - min(x)}$ where x is a vector of all ranks in the merged network and x_i is the current rank for edge i

20.3 Running `seidr compare`

As a minimum, the user needs to provide the translation (`-t, --translate`) and two networks in the binary `SeidrFile` (see *SeidrFiles*) format. This will create a new file (by default "compare.sf") containing the merged network:

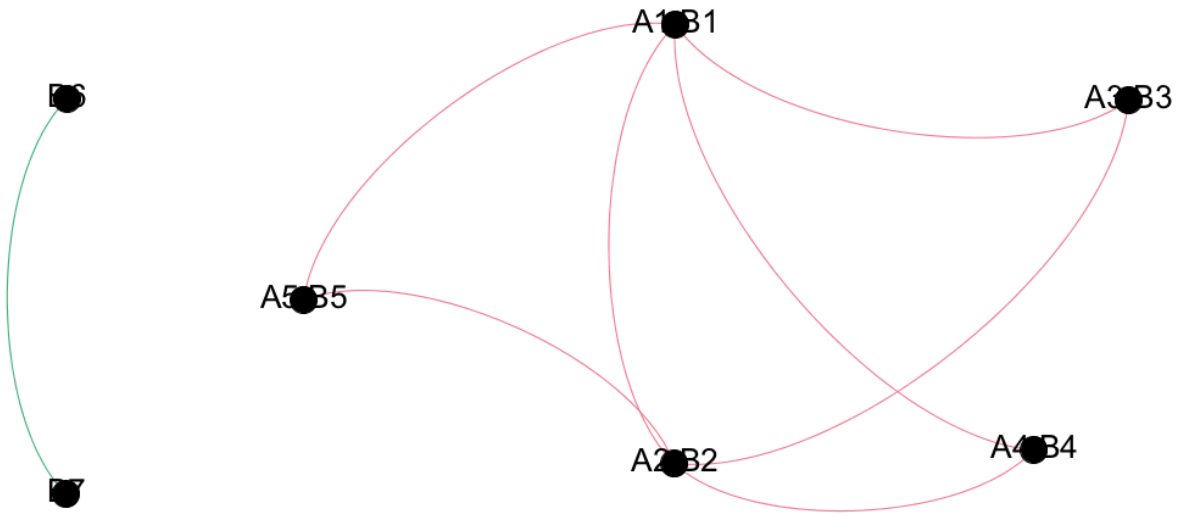
```
seidr compare -t dict.txt net1.sf net2.sf
```

20.4 Output of `seidr compare`

The output of `seidr compare` in its default mode is a merged network. Nodes with overlaps will be comma separated. If e.g. node A1 ine network A matches node B1 in network B, the joined new node will be "A1,B1". The fourth column of the merged network contains important metadata for the edges:

- _Flag_: The flag indicates whether the edge was found in both networks (0), only in the first network (1) or only in the second network (2).
- _Rank_A_: This is the original rank of the edge in network A. If it was not present in network A, its rank will be 0.
- _Rank_B_: Analogous to _Rank_A_.

Source	Target	Type	Weight;Rank	Flag;Rank_A;Rank_B
A2,B2	A1,B1	Undirected	0;15	0;7;8
A3,B3	A1,B1	Undirected	0.307692;1	0;5;6
A3,B3	A2,B2	Undirected	0.769231;5	0;2;3
A4,B4	A1,B1	Undirected	1;2	0;1;1
A4,B4	A2,B2	Undirected	0.461538;9	0;4;5
A5,B5	A1,B1	Undirected	0.615385;7	0;3;4
A5,B5	A2,B2	Undirected	0.153846;13	0;6;7
B7	B6	Undirected	1;2	2;0;2



20.5 Running `seidr compare -n`

If you are interested in nodes, rather than edges, you can run `seidr compare` with the `-n` option. This will create output describing whether a node had at least one edge in network A (1), network B (2) or both (0):

```
seidr compare -n -t dict.txt net1.sf net2.sf
```

20.6 Output of `seidr compare -n`

The output of `seidr compare -n` will be 2 columns as TAB delimited text written to `stdout` indicating whether a node had at least one edge in network A (1), network B (2) or both (0):

```
A1,B1 0
A2,B2 0
A3,B3 0
A4,B4 0
A5,B5 0
B6  2
B7  2
```

CHAPTER 21

Running seidr with nextflow

If you are looking for a convenient way to run seidr, you can consider running it off nextflow . We provide an example configuration (`nextflow.config`) and nextflow pipeline `vala.nf` in the `nextflow` directory of the project root.

Currently, running on a local machine, and on a SLURM cluster are supported. Modify the relevant entries in `nextflow.config` and then run:

```
nextflow run vala.nf
```

CHAPTER 22

Using multiple processors to infer networks

A number of computationally intensive network inference algorithms in `seidr` are written using a hybrid MPI/OpenMP approach. This allows for sahred memory parallelism on a single computer or across many nodes in a cluster. Some inference algorithms in `seidr` have been run on hundreds of CPUs across many nodes in a high performance compute cluster.

22.1 Running in OMP mode

By default, if your computer has multiple CPU cores availble, `seidr` will use as many as it can. If the subprogram has parallel processing support, you can control the extent of the parallelization with the `-O, --threads` option.

Example:

```
# Use all available threads by default:
seidr import ...

# Use two threds
seidr import -O 2 ...

# Use environment variables to control the number of threads
export OMP_NUM_THREADS=2
seidr import ..
```

22.2 Running in MPI mode

By default all inference algorithms will use all cores to process data. Let's use `CLR` as an example:

```
mi -m CLR -i expr_mat.tsv -g genes.txt
```

This will spawn eight compute threads (on my laptop) to process the data. In order to control the allocated number of CPUs, we can use the `-O` flag of the `mi` program:

```
mi -O 4 -m CLR -i expr_mat.tsv -g genes.txt
```

This will use 4 compute threads.

If we want to use multiple nodes, we can use we can run the same command as a child of the `mpirun` program. You should first define a hostfile.:

```
mpirun -hostfile myhostfile.cfg mi -m CLR -i expr_mat.tsv -g genes.txt
```

This will spawn a distributed version of the MI inference, running the maximum amount of OpenMP threads. You can combine `mpirun` and the program's `-O` argument to control the number of compute threads each MPI worker spawns.

A special note on MPI rank order: the highest memory node on the cluster you are using should always be rank 0. If there are any high memory tasks, Seidr will assign them to this MPI worker.

For more info on running MPI jobs (including running them on several nodes), please refer to the OpenMPI webpage

22.3 The batchsize argument

All MPI enabled inference algorithms in `seidr` have a `--batch-size` argument. This is the number of genes a compute thread will process at once before requesting more from the master thread. Lower batch sizes will lead to more time spent in I/O operations and more temporary files, but setting it too high might leave compute threads without work for portions of the run. A good rule of thumb is to set this to $\frac{n_{genes}}{n_{nodes}}$. As an example, if I am estimating the network for 25,000 genes using a five nodes, I set `--batch-size` to $\frac{25000}{5} = 5000$. In general, it is safe to let `seidr` decide on the batch size.

CHAPTER 23

SeidrFiles

23.1 Introduction

Seidr employs it's own file format (called SeidrFile) to store network data. This is done to increase performance, as SeidrFiles are:

- Losslessly compressed using bgzip (to save space)
- Ordered in a lower triangular to enable faster algorithms
- Ranked, so that scores can be rank-aggregated

23.2 The SeidrFile header

A SeidrFile has a **header** that keeps information such as the number of edges, nodes, the node names etc. You can view the header of a SeidrFile with the command:

```
seidr view -H <SeidrFile>
```

The output might look something like this:

```
# [G] Nodes: 50
# [G] Edges: 1225
# [G] Storage: Dense
# [G] Algorithms #: 14
# [G] Supplementary data #: 13
# [A] ARACNE
# [A] CLR
# [A] ELNET
# [A] MI
# [A] GENIE3
# [A] LLR
# [A] NARROMI
```

(continues on next page)

(continued from previous page)

```
# [A] PCOR
# [A] PEARSON
# [A] PLSNET
# [A] SPEARMAN
# [A] SVM
# [A] TIGRESS
# [A] irp
# [S] D1
# [S] D2
# [S] D3
# [S] D4
# [S] D5
# [S] D6
# [S] D7
# [S] D8
# [S] D9
# [S] D10
# [S] D11
# [S] D12
# [S] D13
# [R] Version: 0.10.0
# [R] Cmd: seidr aggregate -f -k -m irp aracne.sf clr.sf elnet.sf elranks.sf genie3.
↪sf llr.sf narromi.sf pcor.sf pearson.sf plsnet.sf spearman.sf svm.sf tigress.sf
# [N] G1
# [N] G2
# [N] G3
# [N] G4
# [N] G5
# [N] G6
```

23.3 The SeidrFile body

In the main body of a SeidrFile, we store the edges of a network. Specifically, for each edge, we have at least four columns:

- Source: For directed edges, this is the originating node, for undirected edges, this is simply one of the partners
- Target: For directed edges, this is the destination node, for undirected edges, this is simply the other partner
- Type: Undirected if the node is undirected, Directed otherwise
- X_score;X_rank: This column holds the original score for algorithm "X" as well as its computed rank.

Besides these four mandatory columns, a SeidrFile can hold any number of additional score/rank columns if it is an aggregated or otherwise processed file and and additional supplementary column that annotates the edge with extra information. To view the body of a SeidrFile you can use:

```
seidr view <SeidrFile>
```

Here is the output of a simple imported network:

```
G1      G2      Directed        0.004;334084
G3      G1      Directed        0.334;22729.5
G1      G4      Directed        0.071;89307
G4      G2      Directed        0.053;104778
G3      G4      Directed        0.006;282776
```

And one that is a little more complex, with 14 score/rank columns and a supplementary column at the end. In aggregated SeidrFiles, the representative score/rank is always the rightmost (last) score/rank column:

```
G2  G1  Directed  0.288087;1.30856e+06  nan;nan 1.87357;106802  0.004;334084  -0.
↪018736;243746  0.0904447;42007 0.244;37455.5 0.0128741;202752  -0.159435;202751  1.
↪07712e-05;360264  -0.00225177;1.32058e+06 0.152;26168 nan;nan 0.978291;117022 11
```

You might notice the columns with nan:nan as score/rank. Seidr uses nan as a placeholder to denote a missing edge. That means this particular edge (G2 -> G1) was not found in the second and thirteenth algorithms.

23.4 The SeidrFile index

SeidrFiles can be indexed with the command:

```
seidr index <SeidrFile>
```

This will create an index file with the extension .sfi. The index allows us to access edges quickly in a SeidrFile without having to decompress unnecessary data. Some seidr commands therefore need the index. As an example, let's see what happens if we try to pull out a specific edge from a SeidrFile without an index:

```
seidr view -n G1000:G3 <SeidrFile>
[ ERROR   ][ 2018-05-02T21:35:45 ][ seidr ]: SeidrFile index <SeidrFile.sfi> must␣
↪exist when using --nodelist
```

Otherwise, if the index exists:

```
seidr view -n G1000:G3 ../dream_net1/aggregate/aggregated.sf
G1000 G3  Undirected  0.388607;611152 nan;nan nan;nan 0.001;581639  -0.0200038;209560␣
↪0.00623208;1.16541e+06  0.057;174410  0.00177422;752791 -0.0595161;752789 2.76065e-
↪06;1.11154e+06 -0.0432047;834369 0.031;315583  0.0006;123144 0.507107;458113
```

CHAPTER 24

References

CHAPTER 25

Indices and tables

- genindex
- modindex
- search

Bibliography

[Marbach2012] Marbach, D., Costello, J. C., Küffner, R., Vega, N. M., Prill, R. J., Camacho, D. M., … Stolovitzky, G. (2012). Wisdom of crowds for robust gene network inference. Nature Methods, 9(8), 796–804. https://doi.org/10.1038/nmeth.2016

[Kueffner2012] Küffner, R., Petri, T., Tavakkolkhah, P., Windhager, L., & Zimmer, R. (2012). Inferring gene regulatory networks by ANOVA. Bioinformatics, 28(10), 1376–1382. https://doi.org/10.1093/bioinformatics/bts143

[Margolin2006] Margolin, A. A., Nemenman, I., Basso, K., Wiggins, C., Stolovitzky, G., Dalla Favera, R., & Califano, A. (2006). ARACNE: an algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. BMC Bioinformatics, 7 Suppl 1, S7. https://doi.org/10.1186/1471-2105-7-S1-S7

[Faith2007] Faith, J. J., Hayete, B., Thaden, J. T., Mogno, I., Wierzbowski, J., Cottarel, G., … Gardner, T. S. (2007). Large-scale mapping and validation of Escherichia coli transcriptional regulation from a compendium of expression profiles. PLoS Biology, 5(1), e8. https://doi.org/10.1371/journal.pbio.0050008

[Daub2004] Daub, C. O., Steuer, R., Selbig, J., & Kloska, S. (2004). Estimating mutual information using B-spline functions–an improved similarity measure for analysing gene expression data. BMC Bioinformatics, 5, 118. https://doi.org/10.1186/1471-2105-5-118

[Ruyssinck2014] Ruyssinck, J., Huynh-Thu, V. A., Geurts, P., Dhaene, T., Demeester, P., & Saeys, Y. (2014). NIMEFI: Gene regulatory network inference using multiple ensemble feature importance algorithms. PLoS ONE, 9(3). https://doi.org/10.1371/journal.pone.0092709

[Huynh-Thu2010] Huynh-Thu, V. A., Irrthum, A., Wehenkel, L., & Geurts, P. (2010). Inferring regulatory networks from expression data using tree-based methods. PLoS ONE, 5(9), 1–10. https://doi.org/10.1371/journal.pone.0012776

[Zhang2013] Zhang, X., Liu, K., Liu, Z. P., Duval, B., Richer, J. M., Zhao, X. M., … Chen, L. (2013). NARROMI: A noise and redundancy reduction technique improves accuracy of gene regulatory network inference. Bioinformatics, 29(1), 106–113. https://doi.org/10.1093/bioinformatics/bts619

[Guo2016] Guo, S., Jiang, Q., Chen, L., & Guo, D. (2016). Gene regulatory network inference using PLS-based methods. BMC Bioinformatics, 17(1), 545. https://doi.org/10.1186/s12859-016-1398-6

[Schafer2005] Schäfer, J., & Strimmer, K. (2005). A Shrinkage Approach to Large-Scale Covariance Matrix Estimation and Implications for Functional Genomics. Statistical Applications in Genetics and Molecular Biology, 4(1). https://doi.org/10.2202/1544-6115.1175

[Haury2012] Haury, A.-C., Mordelet, F., Vera-Licona, P., & Vert, J.-P. (2012). TIGRESS: Trustful Inference of Gene REgulation using Stability Selection. BMC Systems Biology, 6(1), 145. https://doi.org/10.1186/1752-0509-6-145

[Friedman2010] Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1). https://doi.org/10.18637/jss.v033.i01

[Cohen1973] Cohen, J. (1973). Eta-squared and partial eta-squared in fixed factor anova designs. Educational and Psychological Measurement, 33(1), 107–112. https://doi.org/10.1177/001316447303300111

[Wright2017] Wright, M. N., & Ziegler, A. (2017). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. Journal of Statistical Software, 77(1), 1–17. https://doi.org/10.18637/jss.v077.i01

[Broido2018] Broido, A. D., & Clauset, A. (2018). Scale-free networks are rare. Retrieved from http://arxiv.org/abs/1801.03400

[Coscia2017] Coscia, M., & Neffke, F. M. H. (2017). Network backboning with noisy data. In Proceedings - International Conference on Data Engineering (pp. 425–436). https://doi.org/10.1109/ICDE.2017.100

[Zhong2014] Zhong, R., Allen, J. D., Xiao, G., & Xie, Y. (2014). Ensemble-based network aggregation improves the accuracy of gene network reconstruction. PloS one, 9(11).

[Langfelder2008] Langfelder, P., & Horvath, S. (2008). WGCNA: an R package for weighted correlation network analysis. BMC bioinformatics, 9(1), 559.