Seidr 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.schiffler@umu.se
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 patterns. 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.

1.1 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++:

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

1.2 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.
2.1 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.

2.2 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 triangular matrix will have the result of each comparison exactly once:

<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G3</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>G5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

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):

```
<table>
<thead>
<tr>
<th></th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>G2</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>G3</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>G4</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>G5</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
</tbody>
</table>
```

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.

2.3 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 containing 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
node 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
```

2.4 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 corre-
  lation. 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.

2.5 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.
2.6 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 example:

```
OMP_NUM_THREAD=1 seidr import -f lm.txt -g nodes.txt -F lm ...
```
Aggregating networks into a crowd network

3.1 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 output a mean of ranks.
- `-m top1`: This will output the edge with the highest score (==lowest rank) of all methods
- `-m top2`: This will output 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/
˓→../gener3_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_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.
˓→22058e+06 1.07712e-05;360264nan;nan nan;nan 1.87357;106802 -0.018736;243746 0.152;
˓→26168 0.244;37455.5 0.0904447;42007 0.288087;1.30856e+06 0.176275;129253
```
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.
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.

4.1 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:
4.2 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].

4.3 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.

4.4 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:

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Nodes</th>
<th>Edges</th>
<th>Scale Free</th>
<th>Clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.842</td>
<td>5</td>
<td>5</td>
<td>0.80504672</td>
<td>0.375</td>
</tr>
<tr>
<td>0.841</td>
<td>5</td>
<td>5</td>
<td>0.80504672</td>
<td>0.375</td>
</tr>
<tr>
<td>0.84</td>
<td>5</td>
<td>5</td>
<td>0.80504672</td>
<td>0.375</td>
</tr>
<tr>
<td>0.839</td>
<td>5</td>
<td>6</td>
<td>0.027420548</td>
<td>0.545455</td>
</tr>
<tr>
<td>0.838</td>
<td>5</td>
<td>6</td>
<td>0.027420548</td>
<td>0.545455</td>
</tr>
<tr>
<td>0.837</td>
<td>5</td>
<td>6</td>
<td>0.027420548</td>
<td>0.545455</td>
</tr>
<tr>
<td>0.836</td>
<td>5</td>
<td>6</td>
<td>0.027420548</td>
<td>0.545455</td>
</tr>
<tr>
<td>0.835</td>
<td>7</td>
<td>7</td>
<td>0.93935184</td>
<td>0.545455</td>
</tr>
<tr>
<td>0.834</td>
<td>7</td>
<td>7</td>
<td>0.93935184</td>
<td>0.545455</td>
</tr>
</tbody>
</table>
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
```
6.1 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.

6.2 Basic Build

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

- gcc
- gcc-c++
- gcc-gfortran
- cmake
- git
- boost-devel
- glpk-devel or COIN-OR CLP (see A note on CLP and GLPK)
- armadillo-devel
- zlib-devel

Once the dependencies are satisfied, build with:

\begin{verbatim}
git clone --recursive https://github.com/bschiffthaler/seidr
cd seidr
mkdir build
\end{verbatim}

(continues on next page)
cmake -DCMAKE_BUILD_TYPE=Release ..
make

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

### 6.3 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 ..
```

to the CMake build options.

### 6.4 Building Parallel STL (PSTL)

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

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

to the CMake build options.

### 6.5 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 from source). If you want to build *seidr* with the CLP backend add:

```
cmake -DNARROMI_USE_CLP=ON ..
```
** Please note that it is currently not recommended to run ANOVERENCE due to inconsistencies with the original implementation that we were not able to clarify with the original author **

ANOVERENCE ([Kueffner2012]) employs the $\eta^2$ 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.

## 7.1 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
1.172264 0.0330010 0.3176543 0.3872639 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
0.876173 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*:
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:

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Perturbations</th>
<th>PerturbationLevels</th>
<th>Treatment</th>
<th>DeletedGenes</th>
<th>OverexpressedGenes</th>
<th>Time</th>
<th>Repeat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>3</td>
</tr>
</tbody>
</table>

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 expr_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 |
```

### 7.2 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)
7.2. Running ANOVERENCE for a subset of genes

| G3 | G5   | 0.397973 |
| G4 | G5   | 0.403535 |
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].

8.1 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:

<table>
<thead>
<tr>
<th></th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4254475</td>
<td>0.0178292</td>
<td>0.9079888</td>
<td>0.4482474</td>
<td>0.1723238</td>
<td></td>
</tr>
<tr>
<td>0.4424002</td>
<td>0.0505248</td>
<td>0.8693676</td>
<td>0.4458513</td>
<td>0.1733112</td>
<td></td>
</tr>
<tr>
<td>1.0568470</td>
<td>0.2084539</td>
<td>0.4674478</td>
<td>0.5050774</td>
<td>0.2448833</td>
<td></td>
</tr>
<tr>
<td>1.1172264</td>
<td>0.0030010</td>
<td>0.3176543</td>
<td>0.3872039</td>
<td>0.2537921</td>
<td></td>
</tr>
<tr>
<td>0.9710677</td>
<td>0.0010565</td>
<td>0.3546514</td>
<td>0.4745322</td>
<td>0.2077183</td>
<td></td>
</tr>
<tr>
<td>1.1308856</td>
<td>0.1220468</td>
<td>0.4024654</td>
<td>0.3484362</td>
<td>0.1666139</td>
<td></td>
</tr>
<tr>
<td>1.0648694</td>
<td>0.1405077</td>
<td>0.4817628</td>
<td>0.4748571</td>
<td>0.1826433</td>
<td></td>
</tr>
<tr>
<td>0.8761173</td>
<td>0.0738140</td>
<td>1.0582917</td>
<td>0.7303661</td>
<td>0.0536562</td>
<td></td>
</tr>
<tr>
<td>1.2059661</td>
<td>0.1534070</td>
<td>0.7608608</td>
<td>0.6558457</td>
<td>0.1577311</td>
<td></td>
</tr>
<tr>
<td>1.0067555</td>
<td>0.0789863</td>
<td>0.8036309</td>
<td>0.8389751</td>
<td>0.0883061</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
</tr>
</thead>
</table>
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 0.798215 0.874873
0 0 0.889133 0
0 0 0 0.860645 0.95965
```

### 8.2 Tuning the number of bins and spline degree

Estimating mutual information 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 `--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 `--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.

### 8.3 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 `--targets` file containing these gene names:

```
G3
G4
```

And running it with the `--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
```
8.4 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.
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].

### 9.1 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:

<table>
<thead>
<tr>
<th></th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4254475</td>
<td>0.4424002</td>
<td>1.0568470</td>
<td>1.1172264</td>
<td>0.9710677</td>
<td>1.1393856</td>
</tr>
<tr>
<td>0.0178292</td>
<td>0.0505248</td>
<td>0.2084539</td>
<td>0.0030010</td>
<td>0.0010565</td>
<td>0.1220468</td>
</tr>
<tr>
<td>0.9079888</td>
<td>0.8693676</td>
<td>0.4674478</td>
<td>0.3176543</td>
<td>0.3546514</td>
<td>0.4024654</td>
</tr>
<tr>
<td>0.4482474</td>
<td>0.4458513</td>
<td>0.5050774</td>
<td>0.3872039</td>
<td>0.4745322</td>
<td>0.3484362</td>
</tr>
<tr>
<td>0.1723238</td>
<td>0.1733112</td>
<td>0.2448833</td>
<td>0.2537921</td>
<td>0.2077183</td>
<td>0.1686139</td>
</tr>
<tr>
<td>0.4482474</td>
<td>0.1204070</td>
<td>0.4817628</td>
<td>0.4748571</td>
<td>0.3426433</td>
<td>0.1826433</td>
</tr>
<tr>
<td>0.0178292</td>
<td>0.0030010</td>
<td>0.9710677</td>
<td>0.0010565</td>
<td>0.1220468</td>
<td>0.1534070</td>
</tr>
<tr>
<td>0.9079888</td>
<td>0.3176543</td>
<td>0.5050774</td>
<td>0.3872039</td>
<td>0.4745322</td>
<td>0.3484362</td>
</tr>
<tr>
<td>0.4482474</td>
<td>0.3176543</td>
<td>0.5050774</td>
<td>0.3872039</td>
<td>0.4745322</td>
<td>0.3484362</td>
</tr>
<tr>
<td>0.1723238</td>
<td>0.2537921</td>
<td>0.2448833</td>
<td>0.2537921</td>
<td>0.2077183</td>
<td>0.1686139</td>
</tr>
<tr>
<td>0.4482474</td>
<td>0.3426433</td>
<td>0.1826433</td>
<td>0.1686139</td>
<td>0.1534070</td>
<td>0.1006755</td>
</tr>
<tr>
<td>0.1723238</td>
<td>0.1826433</td>
<td>0.1006755</td>
<td>0.0883061</td>
<td>0.0789863</td>
<td>0.8036309</td>
</tr>
<tr>
<td>0.4482474</td>
<td>0.3426433</td>
<td>0.1826433</td>
<td>0.1686139</td>
<td>0.1534070</td>
<td>0.1006755</td>
</tr>
<tr>
<td>0.1723238</td>
<td>0.1826433</td>
<td>0.1006755</td>
<td>0.0883061</td>
<td>0.0789863</td>
<td>0.8036309</td>
</tr>
<tr>
<td>0.4482474</td>
<td>0.3426433</td>
<td>0.1826433</td>
<td>0.1686139</td>
<td>0.1534070</td>
<td>0.1006755</td>
</tr>
<tr>
<td>0.1723238</td>
<td>0.1826433</td>
<td>0.1006755</td>
<td>0.0883061</td>
<td>0.0789863</td>
<td>0.8036309</td>
</tr>
<tr>
<td>0.4482474</td>
<td>0.3426433</td>
<td>0.1826433</td>
<td>0.1686139</td>
<td>0.1534070</td>
<td>0.1006755</td>
</tr>
</tbody>
</table>

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
```

### 9.2 Tuning the number of bins and spline degree

Estimating mutual information 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.

### 9.3 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
```
9.4 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*.
This simple executable calculates pearson or spearman correlation from a set of expression data.

10.1 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:

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

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 correlation 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:

<table>
<thead>
<tr>
<th></th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
<th>G6</th>
</tr>
</thead>
<tbody>
<tr>
<td>G3</td>
<td>0.469355</td>
<td>-0.587163</td>
<td>0.127765</td>
<td>0.145338</td>
</tr>
<tr>
<td>G4</td>
<td>-0.587163</td>
<td>0.16474</td>
<td>0.597376</td>
<td>0.0138744</td>
</tr>
<tr>
<td>G5</td>
<td>0.127765</td>
<td>0.597376</td>
<td>0.127765</td>
<td>0.0138744</td>
</tr>
<tr>
<td>G6</td>
<td>0.145338</td>
<td>0.0138744</td>
<td>0.0138744</td>
<td>0.127765</td>
</tr>
</tbody>
</table>

10.2 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.

10.3 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
```
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].

### 11.1 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
1.172264 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*: 
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
```

### 11.2 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).

### 11.3 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:
In this case we will receive an edge list as output:

<table>
<thead>
<tr>
<th>G3</th>
<th>G1</th>
<th>0.894</th>
</tr>
</thead>
<tbody>
<tr>
<td>G3</td>
<td>G2</td>
<td>0</td>
</tr>
<tr>
<td>G3</td>
<td>G4</td>
<td>0.106</td>
</tr>
<tr>
<td>G3</td>
<td>G5</td>
<td>0</td>
</tr>
<tr>
<td>G4</td>
<td>G1</td>
<td>0.894</td>
</tr>
<tr>
<td>G4</td>
<td>G2</td>
<td>0</td>
</tr>
<tr>
<td>G4</td>
<td>G3</td>
<td>0.106</td>
</tr>
<tr>
<td>G4</td>
<td>G5</td>
<td>0</td>
</tr>
</tbody>
</table>

### 11.4 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

### 11.5 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.
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].

### 12.1 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:

<table>
<thead>
<tr>
<th></th>
<th>0.4254475</th>
<th>0.0178292</th>
<th>0.9079888</th>
<th>0.4482474</th>
<th>0.1723238</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gene 1</td>
<td>0.4424002</td>
<td>0.0505248</td>
<td>0.8693676</td>
<td>0.4458513</td>
<td>0.1733112</td>
</tr>
<tr>
<td>Gene 2</td>
<td>1.0568470</td>
<td>0.2084539</td>
<td>0.4674478</td>
<td>0.5050774</td>
<td>0.2448833</td>
</tr>
<tr>
<td>Gene 3</td>
<td>1.1172264</td>
<td>0.0003010</td>
<td>0.3176543</td>
<td>0.3872039</td>
<td>0.2537921</td>
</tr>
<tr>
<td>Gene 4</td>
<td>0.9710677</td>
<td>0.0010565</td>
<td>0.3546514</td>
<td>0.4745322</td>
<td>0.2077183</td>
</tr>
<tr>
<td>Gene 5</td>
<td>1.1393856</td>
<td>0.1220468</td>
<td>0.4024654</td>
<td>0.3484362</td>
<td>0.1686139</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>1.0648694</th>
<th>0.1405077</th>
<th>0.4817628</th>
<th>0.4748571</th>
<th>0.1826433</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gene 1</td>
<td>0.8761173</td>
<td>0.0738140</td>
<td>1.0582917</td>
<td>0.7303661</td>
<td>0.0536562</td>
</tr>
<tr>
<td>Gene 2</td>
<td>1.2059661</td>
<td>0.1534070</td>
<td>0.7608608</td>
<td>0.6558457</td>
<td>0.1577311</td>
</tr>
<tr>
<td>Gene 3</td>
<td>1.0006755</td>
<td>0.0789863</td>
<td>0.8036309</td>
<td>0.8389751</td>
<td>0.0883061</td>
</tr>
</tbody>
</table>

With that, we can run GENIE3:
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
```

### 12.2 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

### 12.3 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
```

### 12.4 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*. 
Narromi is an MI based algorithm that tries to minimize noise in the MI using linear programming. It is published in [Zhang2013].

### 13.1 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:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>0.581267</th>
<th>0.00822935</th>
<th>0.0106747</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.100319</td>
<td>0.00249005</td>
<td>0.0137571</td>
<td>9.62593e-05</td>
<td></td>
</tr>
<tr>
<td>0.116941</td>
<td>0.00249005</td>
<td>0</td>
<td>0.624368</td>
<td>0</td>
</tr>
<tr>
<td>0.00822935</td>
<td>0.0137571</td>
<td>0.50236</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.0106747</td>
<td>9.62593e-05</td>
<td>0.29456</td>
<td>0.199657</td>
<td>0</td>
</tr>
</tbody>
</table>

### 13.2 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.

### 13.3 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:

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G3</td>
</tr>
<tr>
<td>G4</td>
</tr>
</tbody>
</table>

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:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G3</td>
<td>G1</td>
<td>0.116941</td>
</tr>
<tr>
<td>G3</td>
<td>G2</td>
<td>0.00249005</td>
</tr>
<tr>
<td>G3</td>
<td>G4</td>
<td>0.624368</td>
</tr>
<tr>
<td>G3</td>
<td>G5</td>
<td>0</td>
</tr>
<tr>
<td>G4</td>
<td>G1</td>
<td>0.00822935</td>
</tr>
<tr>
<td>G4</td>
<td>G2</td>
<td>0.0137571</td>
</tr>
<tr>
<td>G4</td>
<td>G3</td>
<td>0.50236</td>
</tr>
<tr>
<td>G4</td>
<td>G5</td>
<td>0</td>
</tr>
</tbody>
</table>

### 13.4 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
PCor is an MI based algorithm that tries to minimize noise in the MI using linear programming. It is published in [Schafer2005].

14.1 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
```

### 14.2 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.

### 14.3 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
```
PLSNET uses a partial least squares feature selection algorithm to predict interacting genes. It is published in [Guo2016].

15.1 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:

```
<table>
<thead>
<tr>
<th></th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4254475</td>
<td>0.0178292</td>
<td>0.9079888</td>
<td>0.4482474</td>
<td>0.1723238</td>
<td></td>
</tr>
<tr>
<td>0.4424002</td>
<td>0.0505248</td>
<td>0.8693676</td>
<td>0.4458513</td>
<td>0.1733112</td>
<td></td>
</tr>
<tr>
<td>1.0568470</td>
<td>0.2084539</td>
<td>0.4674478</td>
<td>0.5050774</td>
<td>0.2448833</td>
<td></td>
</tr>
<tr>
<td>1.1172264</td>
<td>0.0030010</td>
<td>0.3176543</td>
<td>0.3872039</td>
<td>0.2537921</td>
<td></td>
</tr>
<tr>
<td>0.9710677</td>
<td>0.0010565</td>
<td>0.3546514</td>
<td>0.4745322</td>
<td>0.2077183</td>
<td></td>
</tr>
<tr>
<td>1.1393856</td>
<td>0.1220468</td>
<td>0.4024654</td>
<td>0.3484362</td>
<td>0.1686139</td>
<td></td>
</tr>
<tr>
<td>1.0648694</td>
<td>0.1405077</td>
<td>0.4817628</td>
<td>0.4748571</td>
<td>0.1826433</td>
<td></td>
</tr>
<tr>
<td>0.8761173</td>
<td>0.0738140</td>
<td>1.0582917</td>
<td>0.7303661</td>
<td>0.0536562</td>
<td></td>
</tr>
<tr>
<td>1.2059661</td>
<td>0.1534070</td>
<td>0.7608608</td>
<td>0.6558457</td>
<td>0.1577311</td>
<td></td>
</tr>
<tr>
<td>1.0006755</td>
<td>0.0789863</td>
<td>0.8036309</td>
<td>0.8389751</td>
<td>0.0883061</td>
<td></td>
</tr>
</tbody>
</table>
```

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:
The output is a square matrix of scores:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10661.7</th>
<th>9103.01</th>
<th>3781.48</th>
<th>8553.33</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1672.03</td>
<td>0</td>
<td>3808.75</td>
<td>4130.81</td>
<td>24318.7</td>
</tr>
<tr>
<td>1</td>
<td>2783.44</td>
<td>6850.63</td>
<td>0</td>
<td>2885.31</td>
<td>23882.1</td>
</tr>
<tr>
<td>2</td>
<td>1683.27</td>
<td>11640.3</td>
<td>4560.34</td>
<td>0</td>
<td>63590</td>
</tr>
<tr>
<td>3</td>
<td>1218.51</td>
<td>20635.5</td>
<td>9127.68</td>
<td>14218.4</td>
<td>0</td>
</tr>
</tbody>
</table>

15.2 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.

15.3 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 **--targets** file containing these gene names:

```
G3
G4
```

And running it with the **--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
```

15.4 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
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].

16.1 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:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4254475</td>
<td>0.0178292</td>
<td>0.9079888</td>
<td>0.4482474</td>
<td>0.1723238</td>
<td></td>
</tr>
<tr>
<td>0.4424002</td>
<td>0.0505248</td>
<td>0.8693676</td>
<td>0.4458513</td>
<td>0.1733112</td>
<td></td>
</tr>
<tr>
<td>1.0568470</td>
<td>0.2084539</td>
<td>0.4674478</td>
<td>0.5050774</td>
<td>0.2448833</td>
<td></td>
</tr>
<tr>
<td>1.1172264</td>
<td>0.0030010</td>
<td>0.3176543</td>
<td>0.3872039</td>
<td>0.2537921</td>
<td></td>
</tr>
<tr>
<td>0.9710677</td>
<td>0.0010565</td>
<td>0.3546514</td>
<td>0.4745322</td>
<td>0.2077183</td>
<td></td>
</tr>
<tr>
<td>1.139856</td>
<td>0.1220468</td>
<td>0.4024654</td>
<td>0.3484362</td>
<td>0.1686139</td>
<td></td>
</tr>
<tr>
<td>1.0648694</td>
<td>0.1405077</td>
<td>0.4817628</td>
<td>0.4748571</td>
<td>0.1826433</td>
<td></td>
</tr>
<tr>
<td>0.8761173</td>
<td>0.0738140</td>
<td>1.0582917</td>
<td>0.7303661</td>
<td>0.0536562</td>
<td></td>
</tr>
<tr>
<td>1.2059661</td>
<td>0.1534070</td>
<td>0.7608608</td>
<td>0.6558457</td>
<td>0.1577311</td>
<td></td>
</tr>
<tr>
<td>1.0006755</td>
<td>0.0789863</td>
<td>0.8036309</td>
<td>0.8389751</td>
<td>0.0883061</td>
<td></td>
</tr>
</tbody>
</table>

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:
16.2 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.

16.3 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:

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:

<table>
<thead>
<tr>
<th></th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
</tr>
</thead>
<tbody>
<tr>
<td>G3</td>
<td>0.4819</td>
<td>0.1774</td>
<td>0.3084</td>
<td>0.7075</td>
<td>0.137</td>
</tr>
<tr>
<td>G4</td>
<td>0.137</td>
<td>0.1418</td>
<td>0.3349</td>
<td>0.675</td>
<td>0.675</td>
</tr>
</tbody>
</table>

16.4 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
Comparing networks of different species with seidr

17.1 Introduction

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

net1.sf:

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Type</th>
<th>Weight;Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>A2</td>
<td>Directed</td>
<td>7.82637e-06;7</td>
</tr>
<tr>
<td>A1</td>
<td>A3</td>
<td>Directed</td>
<td>0.131538;5</td>
</tr>
<tr>
<td>A2</td>
<td>A3</td>
<td>Directed</td>
<td>0.532767;2</td>
</tr>
<tr>
<td>A1</td>
<td>A4</td>
<td>Directed</td>
<td>0.755605;1</td>
</tr>
<tr>
<td>A2</td>
<td>A4</td>
<td>Directed</td>
<td>0.218959;4</td>
</tr>
<tr>
<td>A1</td>
<td>A5</td>
<td>Directed</td>
<td>0.45865;3</td>
</tr>
<tr>
<td>A2</td>
<td>A5</td>
<td>Directed</td>
<td>0.0470446;6</td>
</tr>
</tbody>
</table>

net2.sf:

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Type</th>
<th>Weight;Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>B2</td>
<td>Directed</td>
<td>7.82637e-06;8</td>
</tr>
<tr>
<td>B1</td>
<td>B3</td>
<td>Directed</td>
<td>0.131538;6</td>
</tr>
<tr>
<td>B2</td>
<td>B3</td>
<td>Directed</td>
<td>0.532767;3</td>
</tr>
<tr>
<td>B1</td>
<td>B4</td>
<td>Directed</td>
<td>0.755605;1</td>
</tr>
<tr>
<td>B2</td>
<td>B4</td>
<td>Directed</td>
<td>0.218959;5</td>
</tr>
<tr>
<td>B1</td>
<td>B5</td>
<td>Directed</td>
<td>0.45865;4</td>
</tr>
<tr>
<td>B2</td>
<td>B5</td>
<td>Directed</td>
<td>0.0470446;7</td>
</tr>
<tr>
<td>B6</td>
<td>B7</td>
<td>Directed</td>
<td>0.678865;2</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
</tr>
<tr>
<td>A4</td>
<td>B4</td>
</tr>
<tr>
<td>A5</td>
<td>B5</td>
</tr>
</tbody>
</table>

### 17.2 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 \( i \)

### 17.3 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
```

### 17.4 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 in 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_.

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Type</th>
<th>Weight;Rank</th>
<th>Flag;Rank_A;Rank_B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2,B2</td>
<td>A1,B1</td>
<td>Undirected</td>
<td>0;15</td>
<td>0;7;8</td>
</tr>
<tr>
<td>A3,B3</td>
<td>A1,B1</td>
<td>Undirected</td>
<td>0.307692;1</td>
<td>0;5;6</td>
</tr>
<tr>
<td>A3,B3</td>
<td>A2,B2</td>
<td>Undirected</td>
<td>0.769231;5</td>
<td>0;2;3</td>
</tr>
<tr>
<td>A4,B4</td>
<td>A1,B1</td>
<td>Undirected</td>
<td>1;2</td>
<td>0;1;1</td>
</tr>
<tr>
<td>A4,B4</td>
<td>A2,B2</td>
<td>Undirected</td>
<td>0.461538;9</td>
<td>0;4;5</td>
</tr>
<tr>
<td>A5,B5</td>
<td>A1,B1</td>
<td>Undirected</td>
<td>0.615385;7</td>
<td>0;3;4</td>
</tr>
<tr>
<td>A5,B5</td>
<td>A2,B2</td>
<td>Undirected</td>
<td>0.153846;13</td>
<td>0;6;7</td>
</tr>
<tr>
<td>B7</td>
<td>B6</td>
<td>Undirected</td>
<td>1;2</td>
<td>2;0;2</td>
</tr>
</tbody>
</table>
17.5 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
```
17.6 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
```
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:

```bash
nextflow run vala.nf
```
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 shared 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.

19.1 Running in OMP mode

By default, if your computer has multiple CPU cores available, 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 theds seidr import -O 2 ...
    # Use environment variables to control the number of threads export OMP_NUM_THREADS=2 seidr import ..

19.2 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:
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.

### 19.3 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 $\frac{n_{\text{genes}}}{n_{\text{nodes}}}$.

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.
20.1 Introduction

Seidr employs its 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

20.2 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
```
20.3 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 any 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:

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Type</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>G2</td>
<td>Directed</td>
<td>0.004;334084</td>
</tr>
<tr>
<td>G3</td>
<td>G1</td>
<td>Directed</td>
<td>0.334;22729.5</td>
</tr>
<tr>
<td>G1</td>
<td>G4</td>
<td>Directed</td>
<td>0.071;89307</td>
</tr>
<tr>
<td>G4</td>
<td>G2</td>
<td>Directed</td>
<td>0.053;104778</td>
</tr>
<tr>
<td>G3</td>
<td>G4</td>
<td>Directed</td>
<td>0.006;282776</td>
</tr>
</tbody>
</table>
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 | −0.18736;243746 | 0.9094447;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 |

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.

### 20.4 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;752789 −0.0595161;752789 2.76065e−06;1.11154e+06 −0.0432047;834369 0.031;315583 0.0006;123144 0.507107;458113
```
CHAPTER 22

Indices and tables

• genindex
• modindex
• search
Bibliography


