Inference in Bayesian Networks with R package BayesNetBP

Han Yu, Rachael Hageman Blair

2017-06-08

Contents

1 Introduction

The Bayesian Network Belief Propagation (BayesNetBP) package was developed in the R programming language (https://www.r-project.org/), for probabilistic reasoning in Probabilistic Graphical Models (PGMs) known as Bayesian Networks (BNs). Implementation of the belief propagation is based on the work by Cowell (2005). If you would like to use BayesNetBP in your publications, please cite the package as

> Han Yu, Moharil Janhavi, Rachael Hageman Blair. "BayesNetBP: An R package for probabilistic reasoning in Bayesian Networks". *Submitted*.

Bayesian networks are a class of PGMs that convey directed dependencies between variables (nodes) in the network. These dependencies can be read from the graph as conditional probabilities and are factored terms in a compact representation of the joint probability distribution for the variables in the network Lauritzen (1996), Koller and Friedman (2009). A BN is a Directed Acyclic Graph (DAG), which ensures there is no feedback in the network, and is required for the factorization of the joint distribution. A BN for a set of random variables, $D = \{X_1, X_2, \ldots X_n\}$, obeys the *Markov condition*, which states that each variable, X_i, is independent of its ancestors, given its parents in the graphs, G. Under these assumptions, the structure of a BN encodes conditional independence relationships:

$$P\left(X_1, X_2, \ldots, X_n\right) = P(G) \prod_{i=1}^{n} P\left(X_i \mid \text{pa}(X_i), \Theta_i\right), \tag{1}$$

where $P(G)$ is the prior distribution over the graph G, $\text{pa}(X_i)$ are the parent nodes of child X_i, and Θ_i denotes the parameters of the local probability distribution.

The BN described in Equation 1 can be constructed directly from the relationships depicted in the DAG (Directed Acyclic Graph). Specifically, the building blocks convey the conditional probability of a child node given its parents, $P\left(X_i \mid \text{pa}(X_i), \Theta_i\right)$. This parent-child relationship is often referred to as a *local model*. For

purely continuous or purely discrete BNs, these local model are often defined using Gaussian regressions and Conditional Probability Table (CPTs), respectively. However, additional modeling assumptions are necessary in the network and local models when there is a mixture of discrete and continuous nodes.

Conditional Gaussian Bayesian Networks (CG-BNs) accommodate a mixture of discrete and continuous variables. Following Lauritzen (1996), Lauritzen and Jensen (2001), we denote the set of discrete nodes as, Δ, and the set of continuous nodes as Γ. For a continuous variable $Y = X_j \in \Gamma$, we define a local model as a conditional Gaussian regression on states of the discrete parents I of Y:

$$L\left(Y \mid I = i, Z = z\right) = N\left(\alpha(i) + \beta(i)^T z, \sigma^2(i)\right),$$

where $\alpha(i)$ is the mean of the regression, β denotes the regression coefficients, $Z \in \Gamma$ are the the continuous parents of Y, and the variance, $\sigma^2(i)$ depends only on the discrete states of the parents. Importantly, in a CG-BN, discrete nodes can be parents of continuous nodes, but not vice-versa.

The package `BayesNetBP` can be seamlessly connected with other tools in R for graphical modeling for the purpose of belief propagation and visualization. The input to `BayesNetBP` is a DAG. The package is flexible in that it can accommodate DAGs learned from different packages in R such as `bnlearn`, `RHugin`, `qtlnet` or a network that is described by an *expert* based on prior knowledge network structure. `BayesNetBP` accommodates networks that are continuous, discrete or a mixture of discrete and continuous variables. However, mixed networks must be structured such that the CG-BN properties regarding discrete nodes preceding continuous nodes, is satisfied.

There are two packages developed in the R programming language that can be used for belief propagation, `RHugin` and `gRain`. `RHugin` can be used for network inference and belief propagation for CG-BNs, but relies on the commercial software Hugin. While a free demo version of huginlite can be used in connection with `RHugin`, the reasoning and inference is limited to smaller networks and datasets (50 states and 500 cases) for the demo version. The `gRain` package can handle large datasets and networks, but it supports probabilistic reasoning only in purely discrete networks. On the other hand, `BayesNetBP`, not only supports probabilistic reasoning in purely discrete, purely continuous, and CG-BNs, but also provides tools for quantification of distributional changes and visualization. Therefore, the `BayesNetBP` package fills a major gap in the graphical modeling tools available in R. The package is the first open source package to facilitate probabilistic reasoning and novel visualizations in all types of BNs. In the following sections, we present examples that are motivated by problems in statistical genetics. However, we emphasize that the `BayesNetBP` can be used in connection with any application.

2 CG-BN example

The data is from the livers from a MRL/MpJ x SM/J mouse intercross, and consists of gene expression data, genotypes at SNP markers and High Density Lipoprotein (HDL) Leduc et al. (2012). Genes that share a QTL with HDL on chromosome 1 and also relate to enriched categories for lipid metabolism in KEGG and Gene Ontologies were selected Alvord et al. (2007). The filtered data used for the modeling can be found in the `BayesNetBP` package. Within this network, we also consider dichotomizing three of the nodes, which creates a second discrete layer in the CG-BN. This example demonstrates the functions for initialization, reasoning and visualizations in a CG-BN.

2.1 Model initialization

For initialization, a graphNEL object of DAG and a vector specifying node types are required to build the semi-elimination tree. In this example, the vector `node.class` indicates which nodes are discrete (TRUE) and continuous (FALSE). The `ClusterTreeCompile` function builds the graph of semi-elimination tree and get the cluster sets, which are the frame work of the final computational object. The `LocalModelCompile` function estimates the local models from a DAG and a data frame. The columns of the data frame must be named by corresponding node names. The local models computed by `LocalModelCompile` function are

distributed into the semi-elimination tree through the `ElimTreeInitialize` function. After initialization, a `ClusterTree` object will be generated.

```
library(BayesNetBP)
data(liver)
liver$node.class
```

```
##         HDL     Pla2g4a       Nr1i3     Cyp2b10      Ppap2a        Kdsr
##        TRUE       FALSE       FALSE        TRUE       FALSE       FALSE
##       Degs1        Neu1       Spgl1       Apoa2  chr1_42.65  chr1_71.35
##       FALSE       FALSE        TRUE       FALSE        TRUE        TRUE
##  chr1_84.93  chr1_86.65 chr12_30.87
##        TRUE        TRUE        TRUE
```

```
# Build cluster tree
cst <- ClusterTreeCompile(dag=liver$dag, node.class = liver$node.class)
# Obtain local models
models <- LocalModelCompile(data=liver$data, dag=liver$dag, node.class=liver$node.class)
# Initialize
tree.init <- ElimTreeInitialize(tree=cst$tree.graph, dag=cst$dag, model=models,
                                node.sets=cst$cluster.sets, node.class=cst$node.class)
# Propagate discrete compartment
tree.init.p <- PropagateDBN(tree.init)
```

The `ClusterTree` object is not ready for evidence absorption or making queries untill its discrete compartment is propagated. The function `PropagateDBN` will perform the propagation within the discrete compartment of the clustertree object so that the joint distribution tables of all clusters are computed. Now the object is ready for evidence absorption and queries.

```
tree.init@propagated
```

```
## [1] FALSE
```

```
tree.init.p <- PropagateDBN(tree.init)
tree.init.p@propagated
```

```
## [1] TRUE
```

2.2 Entering evidence and making queries

The `GetValue` function can be used to check how the values for each discrete variable are coded. For example, the following code shows the locus Chr1@42.65 has three states, while Spgl1 has two states "High" and "Low". This information is helpful for evidence absorption.

```
GetValue(tree.init.p, "Nr1i3")
```

```
## The node is continuous.
```

```
## NULL
```

```
GetValue(tree.init.p, "chr1_42.65")
```

```
## [1] "1" "2" "3"
```

```
GetValue(tree.init.p, "Spgl1")
```

```
## [1] "High" "Low"
```

The `AbsorbEvidence` function can handle evidence of a numeric value for continuous nodes. For a discrete node, the evidence can be either an observed state (hard evidence) or a likelihood (soft evidence). In the following example, these three kinds of evidence: Nrli3 is observed with 1, Chr1@42.65 is observed with state 1, and the likelihood of Spgl1 being High is 0.9, and being Low is 0.2. There three pieces of information can enter the model simultaneously throught the `AbsorbEvidence` function. The absorbed information in the model can also be accessed.

```
tree.post <- AbsorbEvidence(tree.init.p, c("Nr1i3", "chr1_42.65", "Spgl1"),
                            list(1, "1", c(High=0.9, Low=0.2)))
tree.post@absorbed.variables
```

```
## [1] "chr1_42.65" "Nr1i3"
```

```
tree.post@absorbed.values
```

```
## $chr1_42.65
## [1] "1"
##
## $Nr1i3
## [1] 1
```

```
tree.post@absorbed.soft.variables
```

```
## [1] "Spgl1"
```

```
tree.post@absorbed.soft.values
```

```
## $Spgl1
## High  Low
##  0.9  0.2
```

The marginal distributions of both continuous and discrete variables can be queried with the function `Marginals`. For a continuous variable, the marginal is a mixture of Gaussian distributions, output as a data frame with three columns of sub-population probabilities, means and variances. For a discrete variable, the marginal is a named vector of probabilities. The function `SummaryMarginals` computes means and standard deviations for continuous variables as well as returns the number of sub-populations. In the following example, the marginals of both Ppap2a and Neu1 comprise of 108 Gaussian sub-populations.

```
marg <- Marginals(tree.post, c("HDL", "Ppap2a", "Neu1", "chr1_71.35"))
marg$marginals$HDL
```

```
##      High       Low
## 0.2524564 0.7475436
```

```
SummaryMarginals(marg)
```

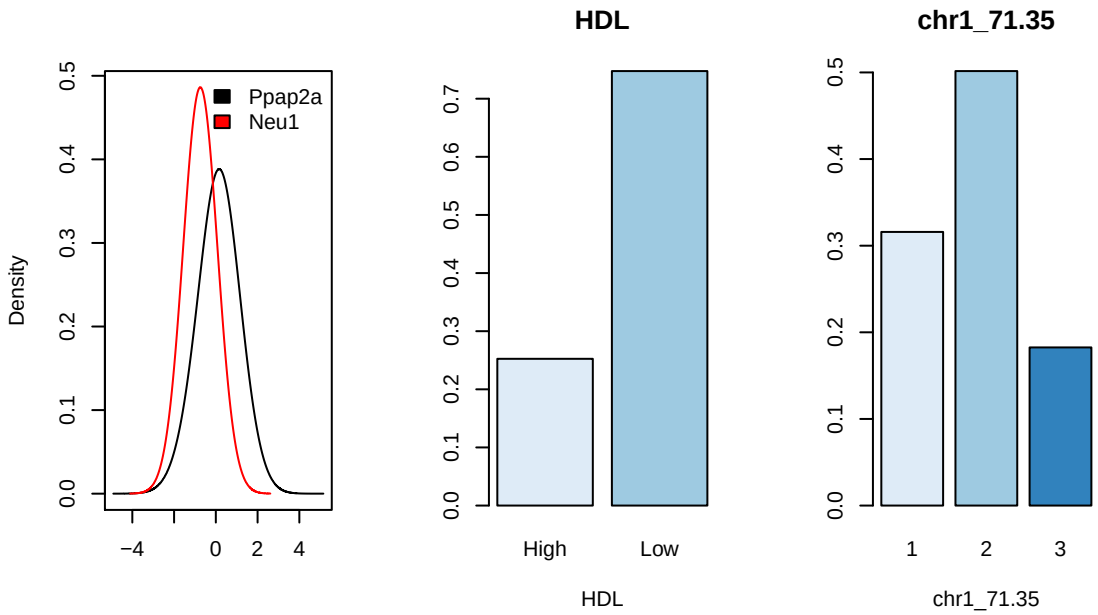
```
##              Mean        SD   n
## Ppap2a  0.1204055 1.0061905 108
## Neu1   -0.7446580 0.6710785 108
```

```
head(marg$marginals$Ppap2a)
```

```
##           prob         mu        sd
## 1 6.514238e-03  0.1466045 1.1614251
## 2 9.852202e-03 -0.6530988 0.8787452
## 3 9.704217e-04 -1.1213652 0.9937244
## 4 3.341952e-03  0.1476780 1.1888417
## 5 2.277200e-03 -0.6808841 0.8862256
## 6 2.432649e-05 -1.3578320 0.9972592
```

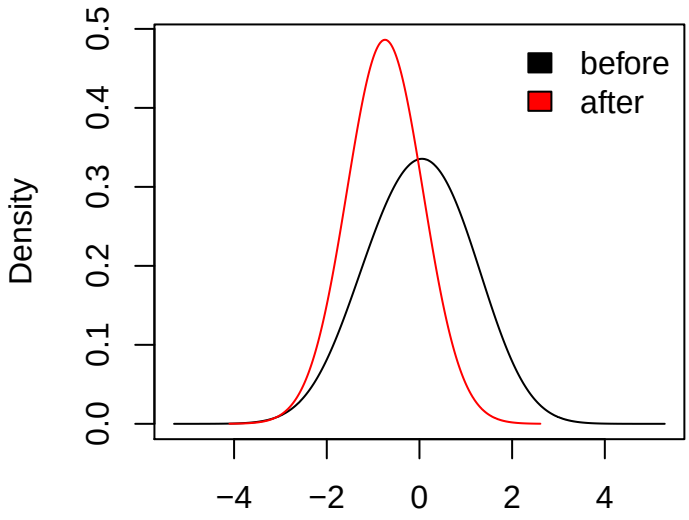
`BayesNetBP` can be used to visualize marginal distributions of both discrete and continuous nodes with the `PlotMarginals` function. The function outputs multiple marginal distributions simultaneously. Marginals of continuous nodes are shown as density plots, while those of discrete nodes as barplots.

```
PlotMarginals(marg)
```



The following codes compare the marginal distributions of Neu1 before and after absorbing the above evidence.

```
var <- "Neu1"
marg.2 <- Marginals(tree.post, var)
marg.1 <- Marginals(tree.init.p, var)
mgns <- list(marg.1$marginals[[1]], marg.2$marginals[[1]])
names(mgns) <- c(var, var)
types <- c(marg.1$types[1], marg.2$types[1])
marg <- list(marginals=mgns, types=types)
PlotMarginals(marg, groups=c("before", "after"))
```



The function `FactorQuery` can provide the joint distribution of any combination of factors, as well as conditional distributions of all discrete variables. In the following example, the joint distrbution of HDL and Cyp2b10, and the conditional distribution of HDL, are computed.

```
# query joint distribution for HDL and Cyp2b10
FactorQuery(tree.post, c("HDL", "Cyp2b10"), mode="joint")
```

```
##    HDL Cyp2b10       prob
## 1 High    High 0.18567653
## 2 High     Low 0.06677985
## 3  Low    High 0.52273666
## 4  Low     Low 0.22480697
```

```
# query joint or conditional distributions for HDL
FactorQuery(tree.post, c("HDL"), mode="conditional")
```

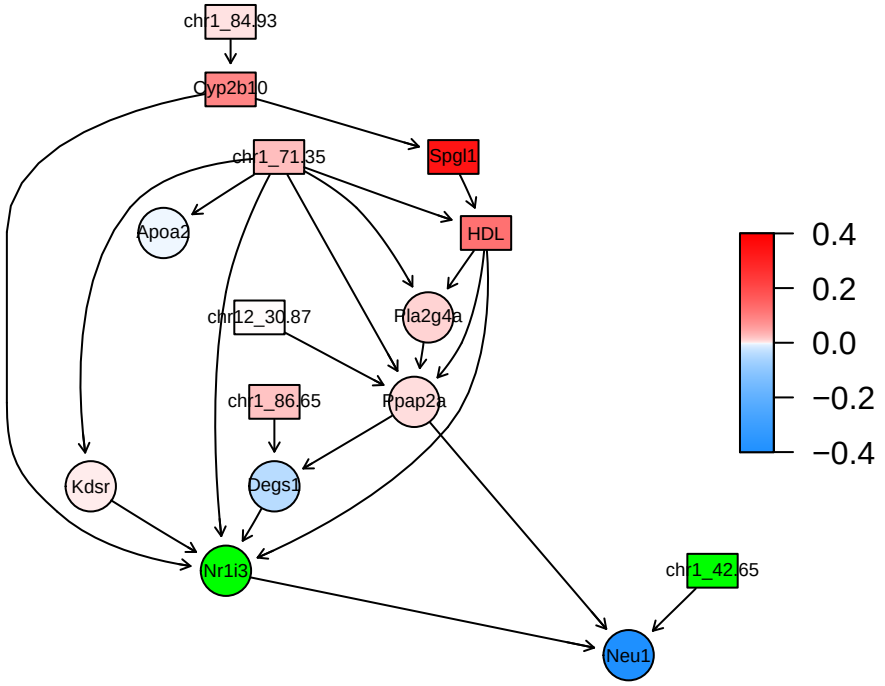
```
##     HDL Spgl1 chr1_71.35       prob
## 1  High  High          1 0.36725775
## 2   Low  High          1 0.63274225
## 3  High  High          2 0.20257483
## 4   Low  High          2 0.79742517
## 5  High  High          3 0.04563497
## 6   Low  High          3 0.95436503
## 7  High   Low          1 0.59305462
## 8   Low   Low          1 0.40694538
## 9  High   Low          2 0.42154566
## 10  Low   Low          2 0.57845434
## 11 High   Low          3 0.17304262
## 12  Low   Low          3 0.82695738
```

2.3 Visualization

The shift of marginals between two models can also be visualized using `PlotCGBN` function. This function takes input of two `ClusterTree` objects, computes the signed and symmetric KL divergence between marginals for each variable, and outputs a graph whose nodes are colored accordingly. Figure 5 shows an example in which Nr1i3 and Chr1@42.65 have absorbed hard evidence (green). The changes of the marginals for the other nodes are quantified using a signed and symmetric KL divergence. They are depicted on the network with a colorbar representaion. Note that the discrete nodes will only change in one direction. For the continuous nodes, red indicates an increase in mean (activation), and blue a decrease in mean (inhibited). This function also returns the signed symmetric KL divergence for each node.

```
PlotCGBN(tree.init.p, tree.post, fontsize = 32)
```

```
##           HDL         Spgl1       Cyp2b10   chr12_30.87    chr1_86.65
##  0.1307547416  0.3534706656  0.0925031249  0.0001042556  0.0245952583
##    chr1_84.93    chr1_71.35          Neu1         Degs1         Ppap2a
##  0.0057804993  0.0288291851 -0.4275696912 -0.0397243702  0.0067383627
##         Apoa2          Kdsr       Pla2g4a
## -0.0015485380  0.0022800388  0.0124116400
```



In the following example, the `PlotCGBN` function is used to demonstrate how conditional independencies within the network change based upon where evidence has been oberved. The Bayesian network in Figure 5 has an active trail between Spgl1 and Chr1@84.93, so after observing Spgl1, the marginal of chr1@84.93 will change (symmetric KL divergence is 0.0047). However, if Cyp2b10 is also observed (gray), then Spgl1 and Chr1@84.93 become d-separated, and further absorption of evidence on Spgl1 will not change the distribution of Cypb10 (symmetric KL divergence is 0). The exploration of the conditional independencies described can be performed by simply absorbing Cyp2b10 into the orginal model to get `tree.1`, and further absorb Spgl1 into it to get `tree.2`, and finally compare `tree.1` and `tree.2` by `PlotCGBN`.

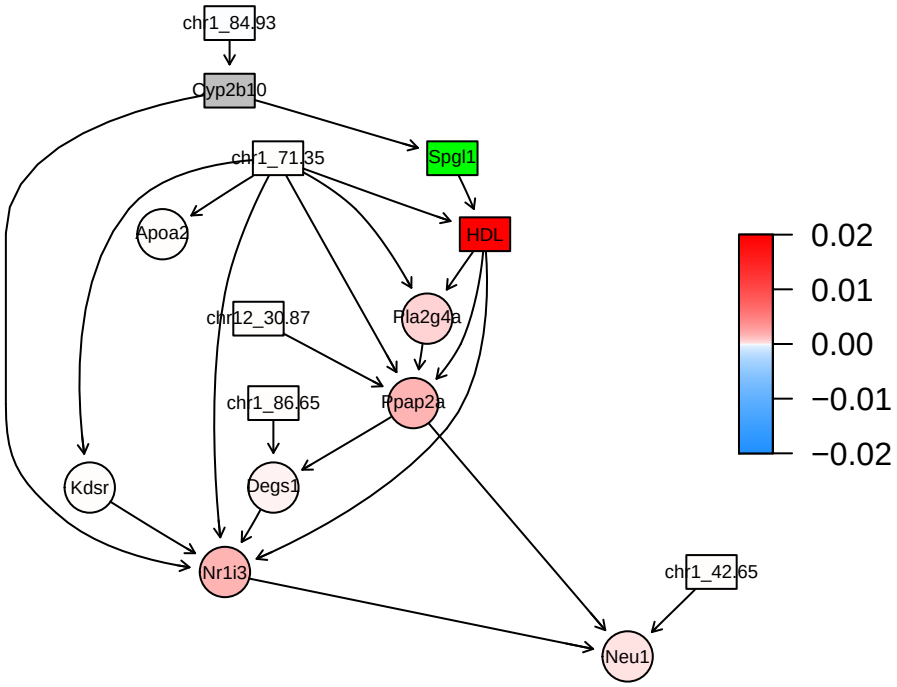
```
tree.1 <- AbsorbEvidence(tree.init.p, c("Cyp2b10"), list("High"))
tree.2 <- AbsorbEvidence(tree.1, c("Spgl1"), list("High"))
```

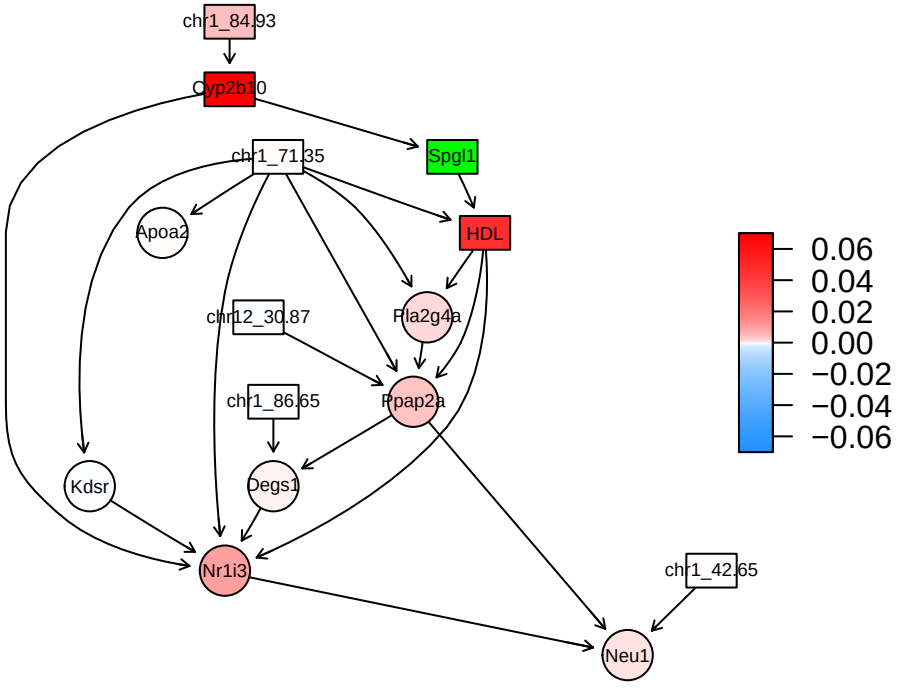
```
tree.3 <- AbsorbEvidence(tree.init.p, c("Spgl1"), list("High"))
PlotCGBN(tree.1, tree.2, fontsize = 32)
```

```
##          HDL  chr12_30.87   chr1_86.65   chr1_84.93   chr1_71.35
## 1.907479e-02 1.299197e-17 4.103860e-17 0.000000e+00 1.198082e-17
##   chr1_42.65         Neu1        Nr1i3        Degs1       Ppap2a
## 4.054185e-17 2.176000e-04 1.621936e-03 6.144299e-05 1.513669e-03
##        Apoa2         Kdsr      Pla2g4a
## 2.209711e-17 6.665141e-18 6.124072e-04
```

```
PlotCGBN(tree.init.p, tree.3, fontsize = 32)
```

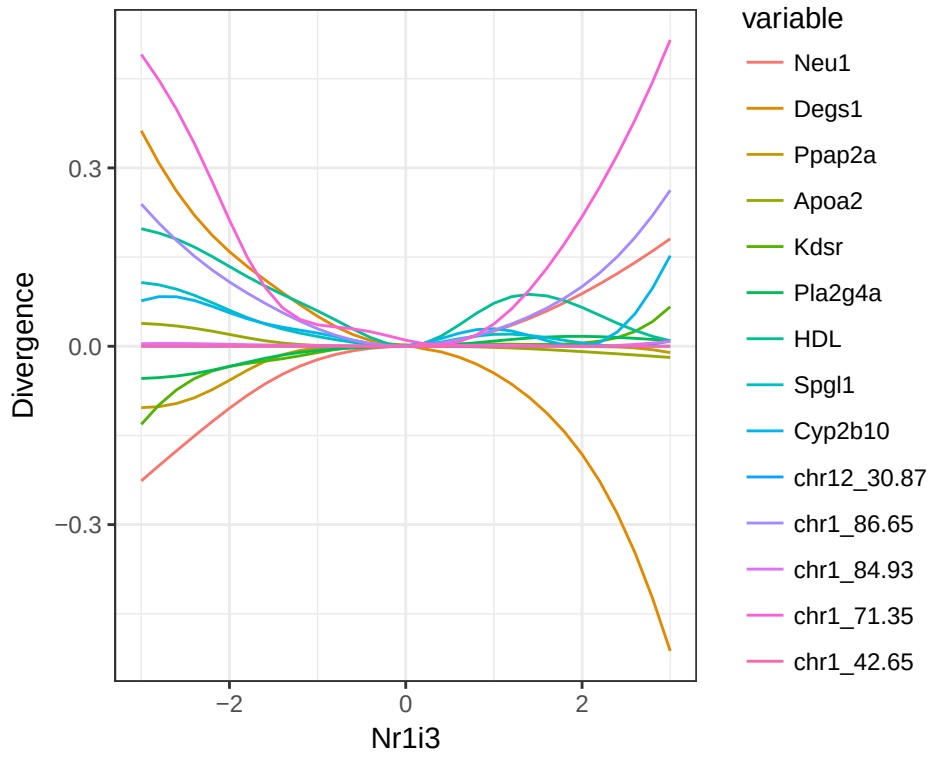
```
##          HDL      Cyp2b10  chr12_30.87   chr1_86.65   chr1_84.93
## 4.813077e-02 7.428658e-02 0.000000e+00 0.000000e+00 4.669436e-03
##   chr1_71.35   chr1_42.65         Neu1        Nr1i3        Degs1
## 1.337859e-17 4.054185e-17 9.954463e-04 1.021268e-02 1.563882e-04
##       Ppap2a        Apoa2         Kdsr      Pla2g4a
## 3.805036e-03 7.237347e-18 0.000000e+00 1.580037e-03
```






```
library(reshape2)
library(ggplot2)
klds <- ComputeKLDs(tree=tree.init.p, var0="Nr1i3",
                    vars=setdiff(tree.init.p@node, "Nr1i3"),
                    seq=seq(-3,3,0.2), pbar=FALSE)
```

```
klds.melt <- melt(klds, id="x")
ggplot(data=klds.melt, aes(x=x, y=value, group=variable, colour=variable)) +
  geom_line() +
  ylab("Divergence") +
  xlab("Nr1i3") +
  theme_bw() +
  theme(legend.key = element_blank())
```



3 Discrete BN example

In this example, we examine a completely dicrete network. The yeast dataset is a subset of the widely studied yeast expression dataset comprising of 112 F1 segregants from a cross between BY4716 and RM11-1a strains of Saccharomyces Cerevisiae Brem and Kruglyak (2005). The original dataset consists of expression values reported as log2(sample/reference) for 6,216 genes. The full data can be accessed in Gene Expression Omnibus (GEO) - accession number GSE1990 Barrett et al. (2013). The subset of genes was identified after filtering, linkage analysis and model building. Briefly, 901 expression values mapped to the YeastNet database H. Kim et al. (2013). Linkage analysis was performed on these traits using R/qtl Broman and Sen (2009). 369 genes that had a significant QTL were used as predictors in an elastic net regression model with COX10 as the response variable Zou and Hastie (2005). The optimal shrinkage parameter was identified as 0.086 using a 10-fold cross validation scheme. The resulting model shrunk all but 37 regression coefficients to zero. The 38 genes including COX10 and their 12 SNP markers corresponding to their QTL were included as variables for the graphical model. This set of 38 genes and their corresponding 12 SNP markers were identified and included in the yeast dataset.

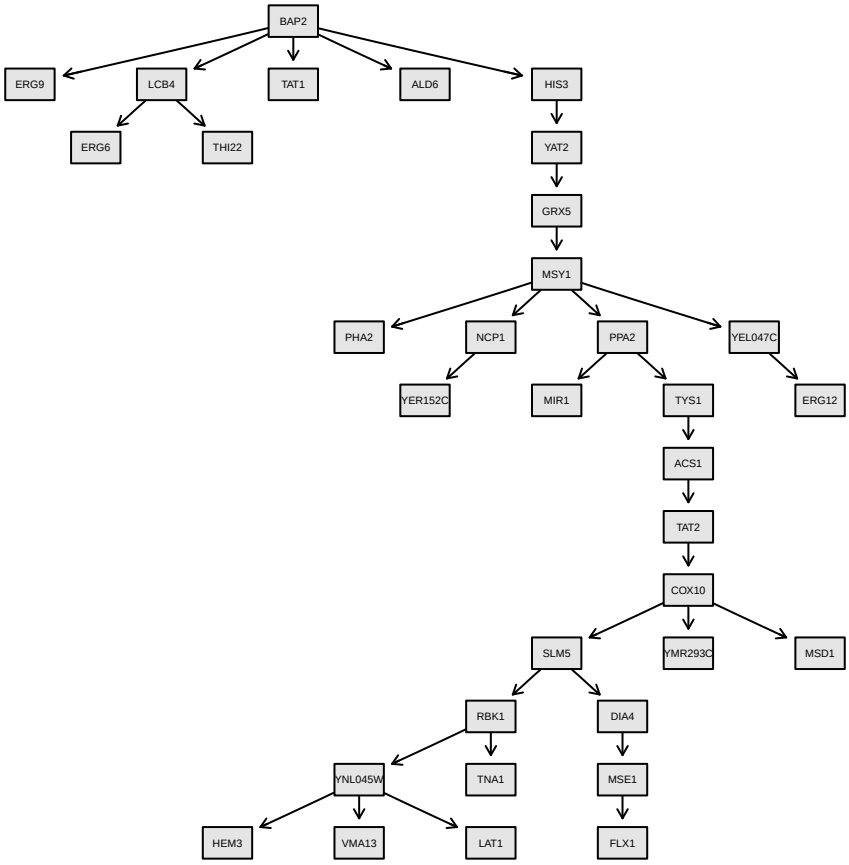
In order to derive discrete nodes for network analysis, the gene expression values were dichotemized at the median. The discrete variables are binary genotype states that indicate the parental strain of origin. To demonstrate compatibility with existing packages, as a starting point, the structure of the BN is learned

using a hill-climbing method in the `bnlearn` package. The modeling assumptions that require genotypes to be upstream of phenotype can be directly encoded using the `blacklist` option.

```
library(bnlearn)
library(igraph)
# learning the network structure
data(yeast)
node.names <- names(yeast)
geno <- node.names[1:12]
pheno <- node.names[13:50]

# get the blacklist
bl <- rbind(expand.grid(geno, geno), expand.grid(pheno, geno))
names(bl) <- c("from", "to")

# fit network
fit <- bnlearn::hc(yeast, blacklist = bl)
dag <- as.graphNEL(fit)
dag.graph <- igraph.from.graphNEL(dag)

# remove isolated nodes
deg <- igraph::degree(dag.graph)
nodes <- names(deg)[deg>0]
dag.graph <- induced_subgraph(dag.graph, nodes)
dag <- igraph.to.graphNEL(dag.graph)
```

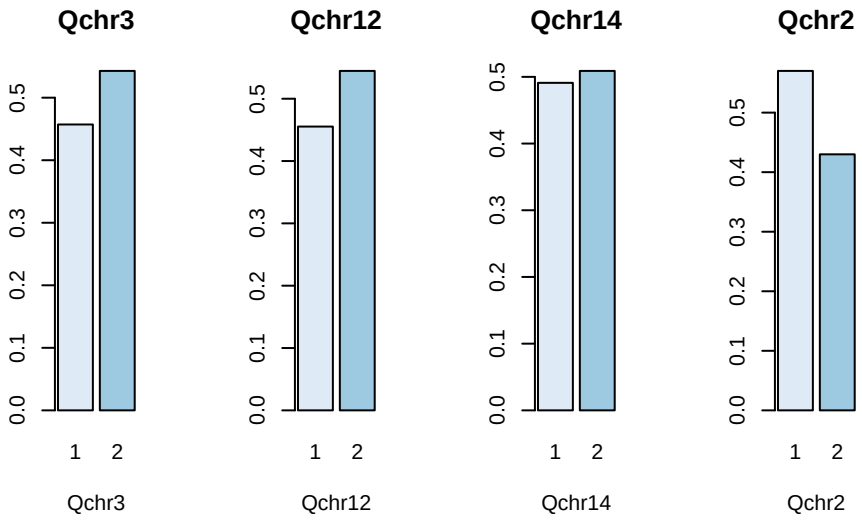
After the network structure being learnt, the models and cluster tree can be built in the same procedure as a CG-BN. `BayesNetBP` brings networks with purely discrete, continuous, or mixed (both discrete and continuous) types of nodes under the same framework, so they can be analyzed conveniently using the identical procedure The following figure shows the structure of constructed semi-elimination tree, which is plotted using the `PlotTree` function. The nodes in the cluster tree plot are named by their corresponding elimination nodes.

```
node.class <- rep(TRUE, length(nodes))
names(node.class) <- nodes
cst <- ClusterTreeCompile(dag=dag, node.class=node.class)
models <- LocalModelCompile(data=yeast, dag=dag, node.class=node.class)
tree.init <- ElimTreeInitialize(tree=cst$tree.graph, dag=cst$dag, model=models,
                                node.sets=cst$cluster.sets, node.class=cst$node.class)
tree.init.p <- PropagateDBN(tree.init)
PlotTree(tree.init.p)
```



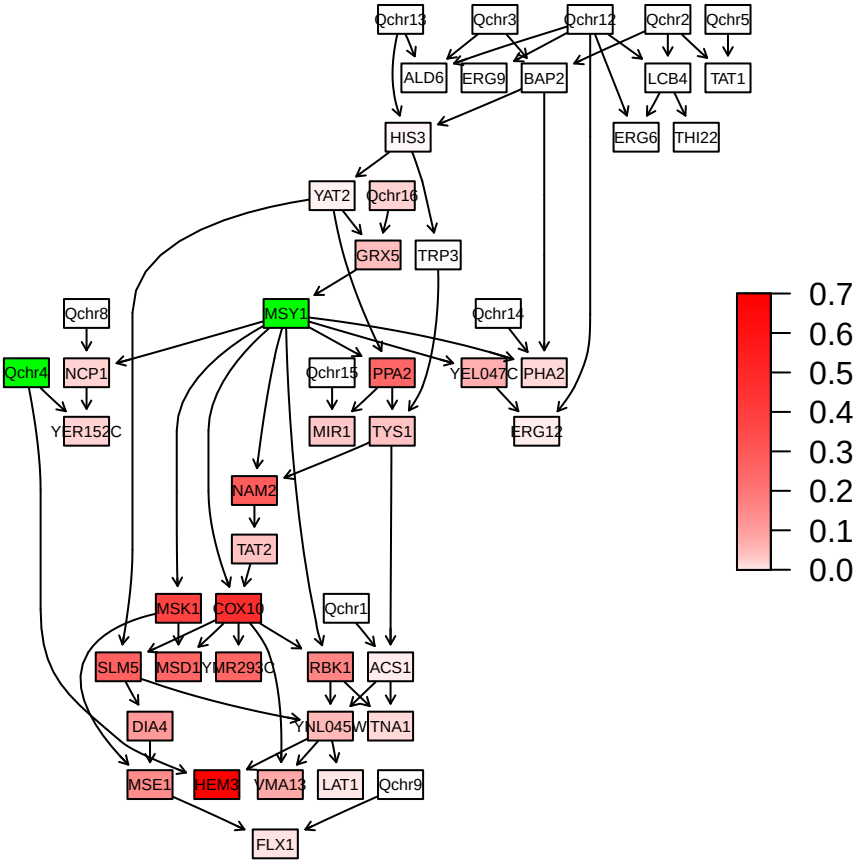
Evidence absorption and marginal computation can be performed in the exactly same manner as in the CG-BN example. The marginals can be visualized using `PlotMarginals`.

```
tree.post <- AbsorbEvidence(tree.init.p, c("MSY1", "Qchr4"), list("1", "2"))
marg.yeast <- Marginals(tree.post, nodes[2:5])
PlotMarginals(marg.yeast)
```



The `PlotCGBN` function can still be used to compare the marginals from two models.

```
div <- PlotCGBN(tree.init.p, tree.post, fontsize = 24)
```



Queries on joint and conditional distributions of factors is also straightforward and can be output in tabular

form. Only conditional distributions of single nodes can be queried. The output is the distribution of the queried node conditional on all its parents.

```
FactorQuery(tree.post, nodes[2:5], mode="joint")
```

```
##    Qchr2 Qchr3 Qchr14 Qchr12       prob
## 1      1     1      1      1 0.05827458
## 2      1     1      1      2 0.06970097
## 3      1     1      2      1 0.06039366
## 4      1     1      2      2 0.07223555
## 5      1     2      1      1 0.06919652
## 6      1     2      1      2 0.08276447
## 7      1     2      2      1 0.07171276
## 8      1     2      2      2 0.08577408
## 9      2     1      1      1 0.04395494
## 10     2     1      1      2 0.05257356
## 11     2     1      2      1 0.04555331
## 12     2     1      2      2 0.05448533
## 13     2     2      1      1 0.05218683
## 14     2     2      1      2 0.06241955
## 15     2     2      2      1 0.05408454
## 16     2     2      2      2 0.06468935
```

```
FactorQuery(tree.post, "ERG9", mode="conditional")
```

```
##   ERG9 Qchr12       prob
## 1   -1      1 0.07843137
## 2    1      1 0.92156863
## 3   -1      2 0.85245902
## 4    1      2 0.14754098
```

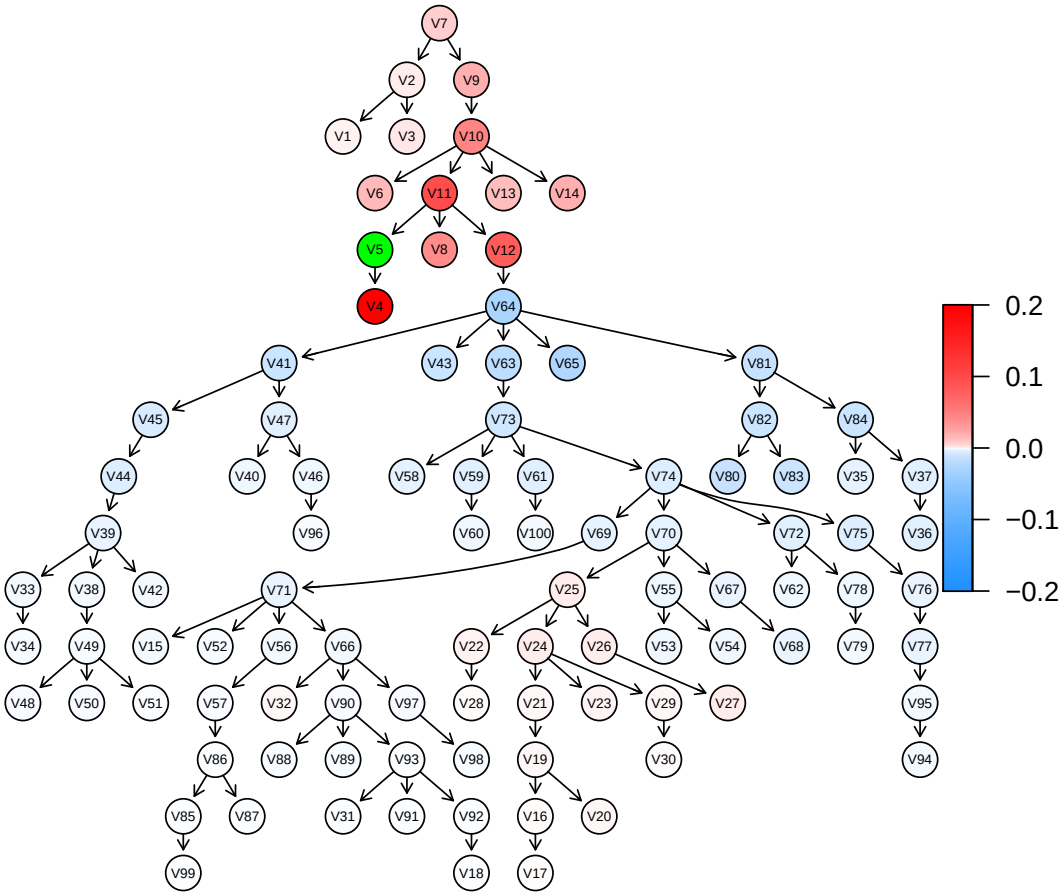
4 Continuous BN example

In this section, we showcase an application of `BayesNetBP` package on purely continuous network without showing the code. The network is constructed based on the 100 genes with highest variances from the NCI60 data set with the `gRapHD` package Abreu, Labouriau, and Edwards (2009). A DAG is induced from the resulted undirected acyclic graph. The following figure visualizes the difference in marginals for models before and after observing $V5 = 2$. The initialization, reasoning and visualization in this setting follows identical steps as previous two examples.

5 BayesNetBP Shiny App

The function `runBayesNetApp` lauches the Shiny App accompanied with this package. The app loads the `toytree` example by default and allows users to load customized `ClusterTree` object. In order to use this feature, a `ClusterTree` object should be built, propagated and named `tree.init.p`, and then saved as a `.RDATA` file. This file can be read in by the app.

The console of `BayesNetBP` Shiny App comprises of three panels. The first part controls the model loading and network layouts. It also allows user to subset the network to faciliate visualization. The `Expand` function can trace the ancestors, descendants, or both, of a selected node in a stepwise manner. The expanded nodes will be colored orange. By clicking `Add to list`, the expanded nodes will be selected and shown purple. The user can then continue selecting additional nodes by using `Expand` and `Add to list` functions. After selecting desired node sets, the user can subset the graph by clicking `Subset`. The nodes in subsetted graph


V7
V2
V9
V1
V3
V10
V6
V11
V13
V14
V5
V8
V12
V4
V64
V41
V43
V63
V65
V81
V45
V47
V73
V82
V84
V44
V40
V46
V58
V59
V61
V74
V80
V83
V35
V37
V39
V96
V60
V100
V69
V70
V72
V75
V36
V33
V38
V42
V71
V25
V55
V67
V62
V78
V76
V34
V49
V15
V52
V56
V66
V22
V24
V26
V53
V54
V68
V79
V77
V48
V50
V51
V57
V32
V90
V97
V28
V21
V23
V29
V27
V95
V86
V88
V89
V93
V98
V19
V30
V94
V85
V87
V31
V91
V92
V16
V20
V99
V18
V17
0.2
0.1
0.0
−0.1
−0.2

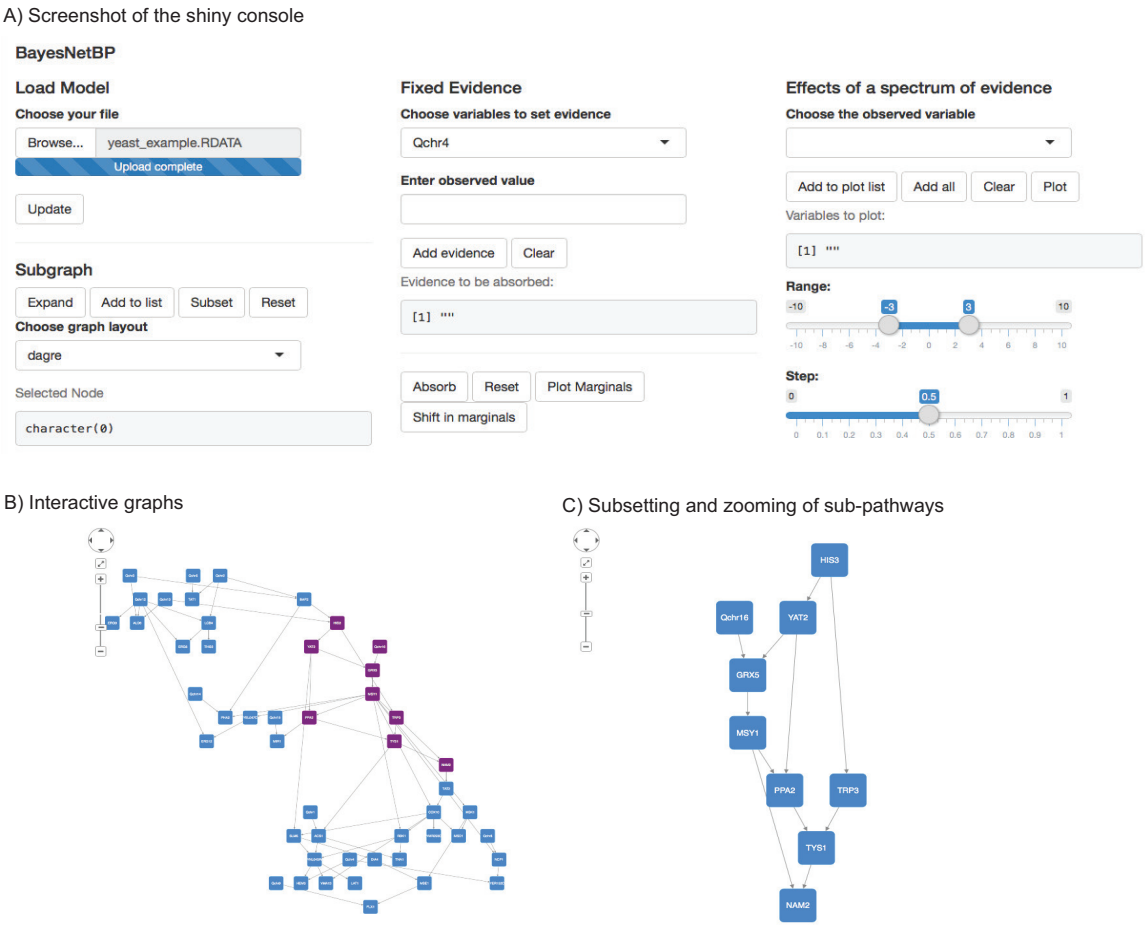


Figure 1: The console of 'BayesNetBP' Shiny App.

retain all properties before subsetting, including their colors and divergence. Other operations can also be performed on the reduced network.

The second panel is used for absorption of fixed and hard evidences. The users can add multiple pieces of evidence to a list and absorb them into the model simultaneously. The nodes with evidence absorbed will be colored green when the absorption is complete. Marginals of the nodes can be quried as density or bar plots by node types. If a set of evidence has been absorbed, the marginals both before and after absorption will be returned to facilitate comparison. To query the marginals, the user can select the node of interest in the graph, and then click `Plot Marginals`. The `Shift in Marginals` function computes the signed and symmetric Kullback-Liebler divergence for all applicable nodes in the network, and colors the nodes in a similar manner as the function `PlotCGBN`.

The function for systematic assessment of variable marginal shifts is provided in the third panel. It allows user to specify which node to absorb the spectrum of evidence in a menu, and to select whose divergence to be calculated by firstly selecting the node on the graph and then clicking `Add to Plot List`. Alternatively, the user can use Add All function to select all applicable nodes into the plotting list. The result is visualized in an interactive plot.

References

Abreu, Gabriel CG de, Rodrigo Labouriau, and David Edwards. 2009. "High-Dimensional Graphical Model Search with GRapHD R Package." *ArXiv Preprint ArXiv:0909.1234*.

Alvord, Gregory, Jean Roayaei, Robert Stephens, Michael W Baseler, H Clifford Lane, and Richard A Lempicki. 2007. "The DAVID Gene Functional Classification Tool: A Novel Biological Module-Centric Algorithm to Functionally Analyze Large Gene Lists." *Genome Biol* 8 (9): 183.

Barrett, Tanya, Stephen E Wilhite, Pierre Ledoux, Carlos Evangelista, Irene F Kim, Maxim Tomashevsky, Kimberly A Marshall, et al. 2013. "NCBI Geo: Archive for Functional Genomics Data Sets?update." *Nucleic Acids Research* 41 (D1). Oxford Univ Press: D991–D995.

Brem, Rachel B, and Leonid Kruglyak. 2005. "The Landscape of Genetic Complexity Across 5,700 Gene Expression Traits in Yeast." *Proceedings of the National Academy of Sciences of the United States of America* 102 (5). National Acad Sciences: 1572–7.

Broman, K. W., and S. Sen. 2009. *A Guide to QTL Mapping with R/Qtl*. Springer.

Cowell, Robert G. 2005. "Local Propagation in Conditional Gaussian Bayesian Networks." *Journal of Machine Learning Research* 6 (Sep): 1517–50.

Kim, Hanhae, Junha Shin, Eiru Kim, Hyojin Kim, Sohyun Hwang, Jung Eun Shim, and Insuk Lee. 2013. "YeastNet V3: A Public Database of Data-Specific and Integrated Functional Gene Networks for Saccharomyces Cerevisiae." *Nucleic Acids Research*. Oxford Univ Press, gkt981.

Koller, Daphne, and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press.

Lauritzen, Steffen L. 1996. *Graphical Models*. Oxford University Press.

Lauritzen, Steffen L, and Frank Jensen. 2001. "Stable Local Computation with Conditional Gaussian Distributions." *Statistics and Computing* 11 (2). Springer: 191–203.

Leduc, Magalie S, Rachael Hageman Blair, Ricardo A Verdugo, Shirng-Wern Tsaih, Kenneth Walsh, Gary A Churchill, and Beverly Paigen. 2012. "Using Bioinformatics and Systems Genetics to Dissect Hdl Cholesterol Levels in an Mrl/Mpj X Sm/J Intercross." *Journal of Lipid Research*. ASBMB, jlr–M025833.

Zou, Hui, and Trevor Hastie. 2005. "Regularization and Variable Selection via the Elastic Net." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67 (2). Wiley Online Library: 301–20.