How To Use catnet Package

Nikolay Balov, Peter Salzman

January 8, 2013

Introduction

The *catnet* package for `R` implements a categorical Bayesian network inference framework. Bayesian networks are graphical statistical models representing causal dependencies between random variables. A Bayesian network has two components: Directed Acyclic Graph (DAG) with nodes the variables of interest and a probability structure given as a set of conditional distributions, one for each node in the graph. Any Bayesian network satisfies the so called *local Markov property*, stating that each node in the network is independent of its non-descendants given its parent nodes. This property implies a factorization of the joint distribution of the node-variables that greatly facilitates the statistical inference.

Two classes of Bayesian networks are among the most used in practice: linear Gaussian networks and categorical ones (also called discrete Bayesian networks). In a linear Gaussian network, the nodes represent continuous variables with Gaussian conditional distributions and the expected value of each node is a linear combination of its parent nodes. Gaussian networks benefit from strong analytical properties and inference methodology. However, their usage is not justified when the linearity and normality assumptions on the observed variables are violated. On the other hand, in a categorical Bayesian network, each node takes value in a fixed set of categories and the conditional distributions are multinomial with no additional parametric constraints. Consequently, when the data of interest is genuinely categorical or the node marginals follow some multi-modal distributions suitable for discretization, the categorical network framework provides greater representation power - one of the reasons to be chosen as a statistical model in *catnet*.

The main goal of the package is to provide tools for inferring categorical Bayesian networks from data based on the maximum likelihood (ML) criterion. The problem of learning Bayesian networks has relatively long history with abundance of literature devoted to its subject. See for example [Heckeman et al.(1995)], [Cooper & Herskovitz(1992)], [Chickering(1996b)], [Friedman et al.(1999)], [Larranaga et al.(1996) and some more recent articles, [Tsamardinos et al.(2003)], [Yaramakala & Margaritis(2005)], [Daly & Shen(2007)]. Although the name Bayesian often implies Bayesian inference, the package follows a frequentist approach without making assumption on the conditional distributions in form of priors, and employs a scoring based approach for network learning. Two main techniques are implemented - finding the best network fitting some data for a predefined node order and stochastic search of optimal networks without constraints on the node order. For a given node order, an efficient exhaustive search via Dynamic Programming (DP) is implemented and the exact MLE solution is given. The approach is similar to that in [Friedman & Koller(2003)] without, however, being Bayesian. The stochastic search in the space of node orders is implemented by employing a Simulated Annealing (SA) algorithm.

A distinct feature of *catnet* is its support of both incomplete data, with some of the records having missing values, and the so called perturbed data. In the latter case, some of the nodes are controlled by breaking the causal influence of their parents. Perturbed data originate naturally from gene expression studies where selected genes are knocked up/down.

The package equips the user not only with structure learning but also with selection, estimation and prediction functions. For example, in *catnet*, one can perform asymptotically consistent model

selection from incomplete or perturbed data, [Balov(2011)]. The motivating goal is to provide more versatile statistical tools for studying networks such as model selection in classes of optimal networks with varying complexity. Despite the diversity and efficiency of the existing algorithms, being strictly structure learning techniques, their selection flexibility is usually limited.

Some attention is paid to the computational issues also. Since the size of the space of networks is super-exponential to the number of nodes, the network learning is inherently a very difficult problem. Nevertheless, manageable inference for moderate size networks of up to several hundred nodes is possible if the node parent sets and overall network complexity are constrained. In addition, *catnet* has parallel processing capabilities allowing for times faster stochastic search.

Although *catnet* is designed as a self-contained package and provides the basic functionality one needs for working with categorical Bayesian networks, some graph related operations such as network visualization are not included. The user may benefit from having installed other packages such as *graph*, for general graph manipulations, and *Graphviz* or *igraph* for graph rendering and visualization.

The authors of *catnet* target reconstruction of gene/protein regulatory networks as a primary application of the package (for overview on this topic see [Sebastiani et al.(2004)] and [Friedman et al.(2000)]), but its functionality is by no means limited for use to bioinformatics only and hopefully will find much larger application scope.

1 Creating and Manipulating Networks

The basic class object in *catnet* package is `catNetwork`, which stands for categorical network. All through the package the object-oriented approach is followed and all classes are defined as $S4$ R-objects. Any `catNetwork` can handle different number of categories for its nodes. Its graph structure, a DAG, describes the relationship between its nodes, while multinomial distributions represent the conditional dependency of the nodes on their parents. For brevity, hereafter we will refer to a `catNetwork` object simply as a network.

Next we present a formal definition of categorical network. Let $\mathcal{V} = \{x_i\}_{i=1}^n$ be a set of n discrete random variables with number of categories $\mathcal{C} = \{c_i\}_{i=1}^n$. A categorical network G with nodes $\mathcal{V}$ is described by its parenthood structure and conditional probabilities. We denote by Π_i the parent set of the i-th node, thus, $\Pi_i \subset \mathcal{V}$ and every $x_j \in \Pi_i$ is a parent of x_i. The conditional probability of x_i given Π_i is denoted $P(x_i|\Pi_i)$ and follows unconstrained categorical distribution. Any permutation Ω of size n defines an order of the nodes $\mathcal{V}$. We say that Ω is consistent with G if

$$\Pi_{\Omega(i)} \subset \{\Omega(1), ..., \Omega(i-1)\}, i = 1, ..., n, \tag{1}$$

thus, the parents of each node appear earlier in the order Ω. Since any categorical network is a DAG and as a such has a consistent node order, the joint probability distribution of G allows the following factorization

$$P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_{\Omega(i)}|x_{\Omega(1)}, ..., x_{\Omega(i-1)}) = \prod_{i=1}^{n} P(x_{\Omega(i)}|\Pi_{\Omega(i)}).$$

Let $D_m = \{X^j\}_{j=1}^m$ be a sample of m observations on variables $\mathcal{V}$. Without loss of generality we assume that for all i an j, $X_i^j \in \{1, 2, ..., c_i\}$. Then the sample-average log-likelihood of G with respect to D_m is

$$l(G|D_m) = \frac{1}{m} \sum_{i=1}^{n} \sum_{j=1}^{m} \log P(X_{\Omega(i)}^j|\Pi_{\Omega(i)}(X^j)), \tag{2}$$

where $\Pi_i(X^j)$ is the realization of the parent set of x_i for the j-th sample X^j. *Catnet* performs network estimation by maximizing l as a function of G.

1.1 Creating New Networks

We start by describing several ways to create a `catNetwork` object. In the usual scenaria the *catnet* is designed for, networks are inferred from data and are created implicitly. There are occasions, however, when the user may want to create a network manually and the package provides such means.

A categorical network can be created explicitly by calling the `cnNew` function. The function takes following arguments: a vector of node names (`nodes`), a list of node categories (`cats`), a list of parents ((`parents`)) and an optional list of conditional probabilities (`probs`). Because of the nested list hierarchy of the probability structure, specifying the probability argument directly can be very elaborated task for large networks. In the following example we create a small network with only three nodes. Note that all inner most vectors in the `probs` argument, such as `(0.4,0.6)`, represent conditional distributions and thus sum to 1. If `probs` parameter is omitted, a random probability structure will be assigned.

```
> library(catnet)
> cnet <- cnNew(
+         nodes = c("a", "b", "c"),
+         cats = list(c("1","2"), c("1","2"), c("1","2")),
+         parents = list(NULL, c(1), c(1,2)),
+         probs = list(        c(0.2,0.8),
+                 list(c(0.6,0.4),c(0.4,0.6)),
+                 list(list(c(0.3,0.7),c(0.7,0.3)),
+                 list(c(0.9,0.1),c(0.1,0.9))))  )
```

1.2 Generating Random Networks

Randomly generated networks can be useful for simulation and evaluation purposes. By calling the `cnRandomCatnet` function, the user may generate a `catNetwork` with random DAG and probability model. The number of nodes, maximum parent size and the number of categories have to be given. All nodes are assigned equal number of categories.

```
> set.seed(123)
> cnet1 <- cnRandomCatnet(numnodes=4, maxParents=2, numCategories=2)
> cnet1

A catNetwork object with  4  nodes,  2  parents,  2  categories,
 Likelihood =  0 , Complexity =  8 .
```

1.3 Inheriting a Graph Object

A `catNetwork` object can be also created by inheriting an existing `graph` object as supported in *graph* package, [Gentleman et al.(2009)]. The latter provides greater number of function for creating and manipulating graphs. A `graph` object can be created directly by specifying its nodes (`myNodes`) and edges (`myEdges`). It contains only a graphical structure description, not a probability one. Then, a `catNetwork` is created by calling the `cnCatnetFromEdges` function.

```
> myNodes<-c("a","s","p","q","r","t","u")
> myEdges<-list(a=list(edges=NULL), s=list(edges=c("p","q")), p=list(edges=c("q")), q=list(edges=c("r"
> cnet2 <- cnCatnetFromEdges(nodes=myNodes, edges=myEdges, numCategories=2)
```

In *catnet* package one is able to import graphs from Simple Interaction Format (SIF) and Bayesian Networks Interchange Format (BIF) files. If a SIF file describes a DAG, which may not be the case since SIF files can describe any graph structure, the graph can be imported by calling the `cnCatnetFromSif` function. In this case a random probability structure is assigned. On the other hand, a BIF file describes both the DAG and probability structures.

```
> #cnet3 <- cnCatnetFromSif(filename)
> #cnet3
```

2 Accessing Network Attributes and Characteristics

There are several functions that give access to the main components of a `catNetwork` object, or more precisely, its slots. Such are the functions `cnNumNodes`, `cnNodes`, `cnEdges`, `cnMatEdges`, `cnParents`, `cnMatParents` and `cnProb`, which are shortly described next.

Of course, all attributes can be accessed using the `@attribute` mechanism, but that opens the possibility of accidental attribute change. Note that, in general, direct manipulation with the network components is not recommended for it may destroy the object integrity.

Functions `cnNumNodes` and `cnNodes` return the number and the list of names, respectively, of network nodes. For each node of a network, for example `cnet1`, one can obtain its parents by calling the `cnParents` function or find its children through `cnEdges` function.

```
> cnNumNodes(cnet1)

[1] 4

> cnNodes(cnet1)

[1] "N1" "N2" "N3" "N4"

> cnEdges(cnet1)

$N2
[1] "N3" "N4"

$N3
[1] "N4"

> cnParents(cnet1)

$N3
[1] "N2"

$N4
[1] "N2" "N3"
```

In addition, the corresponding `cnMatParents` and `cnMatEdges` functions return matrices instead of lists. Specifically, `cnMatParents` returns a binary matrix representing the pairwise node connections, and it is especially useful for comparing networks with the same number of nodes. Alternatively, `cnMatEdges` returns a two-column matrix of ordered pairs encoding the edges with direction from the first to the second.

```
> cnMatParents(cnet1)

   N1 N2 N3 N4
N1  0  0  0  0
N2  0  0  0  0
N3  0  1  0  0
N4  0  1  1  0
```

```
> cnMatEdges(cnet1)

     [,1] [,2]
[1,] "N2" "N3"
[2,] "N2" "N4"
[3,] "N3" "N4"
```

`cnProb` and `cnPlotProb` function provides an access to the complete probability table of a network, which is a recursive list and can be quite large for networks with big parent sets and many categories. Conditional probabilities are reported in the following format. First, node name and its parents are given, then a list of probability values corresponding to all combinations of parent categories (put in brackets) and node categories. In the following example the first node ($N1$) has two categories ($C1$ and $C2$) and no parents, thus two numbers are given, probability 0.68 for the first category and 0.32 for the second. The third node ($N3$) has two categories and one parent ($N2$) and consequently two pairs of probabilities are reported , one for $N2 = C1$ and another for $N2 = C2$.

```
> cnPlotProb(cnet1)

Node[N1], Parents:
[]C1  0.679
[]C2  0.321
 Node[N2], Parents:
[]C1  0.542
[]C2  0.458
 Node[N3], Parents: N2
[ C1]C1  0.103
[ C1]C2  0.897
[ C2]C1  0.855
[ C2]C2  0.145
 Node[N4], Parents: N2 N3
[ C1 C1]C1  0.256
[ C1 C1]C2  0.744
[ C1 C2]C1  0.563
[ C1 C2]C2  0.437
[ C2 C1]C1  0.392
[ C2 C1]C2  0.608
[ C2 C2]C1  0.481
[ C2 C2]C2  0.519
```

An important characteristic of any categorical Bayesian network is its complexity. The complexity is an integer number specifying the number of parameters needed to define the probability structure of the network. For example, the complexity of a network with nodes without parents and two categories per node equals the number of nodes. The complexity of a `catNetwork` object can be obtained by calling the `cnComplexity` function.

```
> cnComplexity(cnet1)

[1] 8
```

The complexity plays a key role in the network estimation and model selection problems.

2.1 Drawing a Network

catnet provides several alternatives for visualizing a network. They are implemented in the function `cnPlot`. If the *igraph* package is installed, a `catNetwork` object will be coerced to a `igraph` object and plotted. Alternatively, `cnPlot` may generate a dot-file, compatible with the external Graphviz software package ([graphvis]). Dot-files can be rendered to a postscript or a pdf files using the `dot` executable from Graphviz or directly visualized by `dotty` or `tcldot`. There is an auxiliary to `cnPlot` function called `cnDot` that generates and saves dot-files specifically.

```
#cnPlot(cnet1)
#cnDot(cnet1, "cnet1.dot")
```

2.2 Topological Node Order

Since any Bayesian network is a DAG, there is a natural order of its nodes such that any node has parents only among the nodes appearing earlier in the order. In fact, a `catNetwork` object may have many compatible orders and the function `cnOrder` returns just one of them. `cnOrder` takes as an input either a `catNetwork` object or a list of parent sets. The next example illustrates its usage.

```
> cnOrder(cnet1)

[1] 1 2 3 4

> cnOrder(cnet1@parents)

[1] 1 2 3 4
```

The topological order is important in the context of network learning and it is another key element in the *catnet*'s search methodology.

2.3 Equivalent Graph Representation

An important result from the theory of Bayesian networks states that all networks with common sets of nodes can be organized in equivalent classes. According to definition, for any two equivalent networks there are probability structures such that their joint probabilities are equal. More on the topic one can find in [Verma & Pearl(1990)] and [Chickering(1996b)]. Function `dag2cpdag` generates the Complete Partially Directed Graph (CPDAG) for a network according to the algorithms given in [Chickering(1996b)]. Note that in a CPDAG some edges are not directed to reflect the freedom of choosing directions without leaving the corresponding equivalent class.

```
> set.seed(456)
> cnet2 <- cnRandomCatnet(numnodes=10, maxParents=3, numCategories=2)
> cnEdges(cnet2)

$N2
[1] "N8"  "N10"

$N4
[1] "N3"

$N6
[1] "N8"

$N8
[1] "N5"
```

```
> pcnet2 <- dag2cpdag(cnet2)

add  N3 -> N4
add  N5 -> N8
add  N10 -> N2

> cnEdges(pcnet2)

$N2
[1] "N8"  "N10"

$N3
[1] "N4"

$N4
[1] "N3"

$N5
[1] "N8"

$N6
[1] "N8"

$N8
[1] "N5"

$N10
[1] "N2"
```

2.4 Comparing Networks

Often, one has two networks with the same nodes and wants to evaluate the difference between them. There are two basic criteria for comparing networks. First, a topological one that compares the graphical structure of the networks and second, a probabilistic one, involving the distributions specified by the networks. *catnet* employs several measures for topological comparison such as the number of true positive/false positive/false negative directed edges, the parent Hamming distance - the number of different elements between the corresponding parent matrices, the number of true positive/false positive/false negative undirected edges (skeleton), and the number of false positive/negative Markov pairs (pairs that have common descendants). Also included is a measure comparing the node order in the networks. The user can compare two networks by calling the `cnCompare` function. It returns a a `catNetworkDistance` object containing the values of the provided measures. More details can be found on `cnCompare`'s help pages.

```
> set.seed(456)
> cnet3 <- cnRandomCatnet(cnNumNodes(cnet2), maxParents=2, numCategories=2)
> cnet3@nodes <- cnet2@nodes
> cnCompare(object1=cnet2, object2=cnet3)

 Edges:
                 TP = 1,
                 FP = 3,
                 FN = 4,
```

```
            F-score = 0.331532,

Hamming:
            (FP+FN) = 7,
                exp = 9,

Skeleton:
                 TP = 1,
                 FP = 3,
                 FN = 4,

Order:
                 FP = 0,
                 FN = 0,

Markov blanket:
                 FP = 0,
                 FN = 1
```

3 Generating Samples and Making Predictions

In addition to the row-sample data representation as often used in statistical practice, *catnet* also allows a column-sample format, the latter being a standard for storing micro-array data. In the latter case, the samples are organized in columns and each row represents a node. The package provides two function for data generation, `cnSample` and `cnSamplePert`. The second one allows the user to generate a perturbed sample, a sample with some of its nodes having fixed, non-random, values. In microbiology, the perturbation techniques is an important tool for inferring causal relationships in regulatory networks.

In the following example we generate a random sample of size 100 from `cnet1` object that have been created earlier. By setting the `output` argument to be "matrix", we obtain a result in `matrix` form. Alternatively, a sample as `data.frame` can be generated.

```
> samples1 <- cnSamples(object=cnet1, numsamples=100, output="matrix")
> dim(samples1)

[1]   4 100

> samples1 <- cnSamples(object=cnet1, numsamples=100, output="frame")
> dim(samples1)

[1] 100   4
```

Another possibility is to generate perturbed samples with fixed, user specified, values for particular nodes. We endow the term `perturbations` with the same meaning as the used in gene experimental analysis - observing steady-state expression levels of selected genes. In presence of genetic perturbations one or more genes are fixed by deletion or over-expression. Perturbed gene experiments are useful tool in studying gene interactions. In non-biological context, if a network node represents some random factor, we can think of the node perturbations as a means to control this factor. To generate perturbed samples, the user may call `cnSamples` function and specify an additional vector argument, `perturbations`, of length the number of nodes. The vector value for each node is either a fixed categorical index from the categorical set of the node, which is carried out unchanged to the output, or zero, marking the node as random one that has to be sampled. In the next example a sample of size 10 is generated with

random first two nodes but perturbed third and fourth nodes that take their first and second category, respectively.

```
> samples2 <- cnSamples(object=cnet1, numsamples=10, perturbations=c(0,0,1,2))
```

For prediction purposes one can use `cnPredict` function with parameters a network object and data. In the data, only the node positions marked as not-available (NA) are predicted. The nodes are assigned categorical values based on the maximum probability criterion. If for example, given a particular instance of its parenthood, a node has three categories with probabilities (0.2, 0.5, 0.3), then the second category will be assigned as its value.

```
>   ## generate a sample of size 12 and set the last 3 nodes as not-available
>   numnodes <- cnNumNodes(cnet2)
>   samples3 <- cnSamples(object=cnet2, numsamples=12, output="matrix")
>   ## predict the last three nodes in 'cnet2' from the rest
>   ## by setting their values in 'samples3' as NA
>   samples3[numnodes-2, ] <- rep(NA, 12)
>   samples3[numnodes-1, ] <- rep(NA, 12)
>   samples3[numnodes, ] <- rep(NA, 12)
>   ## predict the values of the last 3 nodes
>   newsamples <- cnPredict(object=cnet2, data=samples3)
```

4 Learning Network form Data

All existing network learning algorithms such as Grow-Shrink, Incremental Association [Yaramakala & Margaritis(2005)], [Tsamardinos et al.(2003)] and Hill-Climbing [Daly & Shen(2007)], to mention a few, implement either score or constraint-based algorithms to find good solutions. *catnet* implements a global score algorithm and searches for networks fitting the data exhaustively, according to the MLE criterion - in cases when the topological order of the nodes is known, the package provides a function, `cnSearchOrder`, that finds the exact (if is unique) MLE solution for the estimation problem. Alternatively, if the order is not known the user may search the space of orders using the stochastic search function `cnSearchSA`.

4.1 Network Estimation for Given Node Order

As put forward in [Cooper & Herskovitz(1992)], the model search in the space of all networks can be restricted to a smaller space of networks that are consistent with a particular node order. The spaces of networks with fixed node order are not only smaller but more 'regular'. In [Friedman & Koller(2003)], authors evaluate the regularity in terms of posterior distribution of features and also conclude that a search restricted to a particular node ordering does not necessarily preclude good network recovering. Empirical studies also confirm that on order restricted network spaces, the likelihood functions have less local variability, thus discontinuity, which facilitates likelihood based estimations.

Function `cnSearchOrder` is the main computational tool provided by *catnet*. The function implements a Dynamic Programming (DP) algorithm for searching in the space of networks having a node order specified by the user. The result is a set of networks with increasing complexity up to a given maximum value. In other words, each resulting network is exact the maximum likelihood estimator (MLE) for the corresponding complexity. More formally, for given node order Ω and complexity number α, the function finds

$$\hat{G}_{MLE}(\Omega, \alpha) = argmax\{l(G|D_m), \text{ such that } Complexity(G) = \alpha\},$$

where the log-likelihood $l(G|D_m)$ is defined by Eq. 2.

Then the user may proceed by selecting a particular network from the found set of networks by specifying a target complexity or applying some model selection criterion - see the functions `cnFindAIC` and `cnFindBIC` discussed below.

```
> set.seed(789)
> cnet2 <- cnRandomCatnet(numnodes=10, maxParents=2, numCategories=2)
> nodeOrder <- order(runif(cnNumNodes(cnet2)))
> cnet2

A catNetwork object with  10  nodes,  2  parents,  2  categories,
 Likelihood =  0 , Complexity =  11 .

> ## generate a 100-size sample from cnet2
> samples <- cnSamples(object=cnet2, numsamples=100, output="frame")
> netlist <- cnSearchOrder(data=samples, perturbations=NULL, maxParentSet=2, maxComplexity=20,
+         nodeOrder, parentsPool=NULL, fixedParents=NULL)
> ## find the recostructed network with the true complexity
> bnet <- cnFind(netlist, 20)
> bnet

A catNetwork object with  10  nodes,  2  parents,  2  categories,
 Likelihood =  -6.071643 , Complexity =  20 .
```

`cnSearchOrder` has two mandatory parameters: `data` and `maxParentSet`. The data is given as a matrix or data frame of characters and the function extract the categorical set for each node from the data. If `maxComplexity` is not specified, the search is applied to the maximum possible complexity.

In many practical problems some prior information about the network structure is available which the user may want to include in the search process. Such prior information can be obtained from experts in the field of interest or to be based on preceding research. In all of its search functions, *catnet* includes the parameters `parentsPool` and `fixedParents`, that can be used for specifying edge constraints.

The first parameter, `parentsPool`, specifies a set of possible parents for each node and in this way, some nodes can be excluded as potential parents. Additionally, the `fixedParents` parameter specifies edge inclusion rules - the user may assign mandatory parents to some nodes. These two parameters allow a variety of edge constraints to be implemented.

In the next example, we generate a random network with 12 nodes and then search for the best fitting networks that comply with the following requirements: (1) the last node is not a parent to anyone else, and (2) the first two nodes are necessarily parents to all of the rest nodes. The search is restricted to the 'true' node order, the one of the network from which the data is generated, as obtained by `cnOrder(cnet)` function.

```
> set.seed(123)
> nnodes <- 12
> cnet <- cnRandomCatnet(numnodes=nnodes, maxParents=5, numCategories=2)
> norder <- cnOrder(cnet)
> parPool <- vector("list", nnodes)
> for(i in 1:nnodes) parPool[[i]] <- 1:(nnodes-1)
> fixparPool <- vector("list", nnodes)
> for(i in 3:nnodes) fixparPool[[i]] <- c(1,2)
> samples <- cnSamples(cnet, numsamples=200)
> eval <- cnSearchOrder(data=samples, perturbations=NULL, maxParentSet=2, maxComplexity=200,
+         nodeOrder=norder, parentsPool=parPool, fixedParents=fixparPool)
> eval
```

```
 Number of nodes    = 12,
 Sample size        = 200,
 Number of networks = 31
 Processing time    = 0.154

> ## plot likelihood vs complexity for the resulting list of networks
> ## plot(eval@complexity, eval@loglik, xlab="Complexity", ylab = "Log-likelihood", main="Model select
```

4.2 Model Selection with AIC and BIC criteria

The model selection problem in the context of network learning is one of the focuses in *catnet*. As we have mentioned earlier, the package aims to provide flexibility and allows the user to select a network from a list of optimal ones according to the user needs. Methodological details behind the model selection procedures implemented in *catnet* can be found in [Salzman & Almudevar(2006)].

Recall that by calling one of the functions `cnSearchOrder` and `cnSearchSA` one obtains a list of networks, more precisely `catNetwork` objects, such that each one has a unique complexity. From this list one may select a network based on particular criteria such as AIC and BIC - the likelihood alone is not enough to make a selection. In the next example both AIC and BIC criteria are applied and the complexities of the selected networks are marked on the model selection curve.

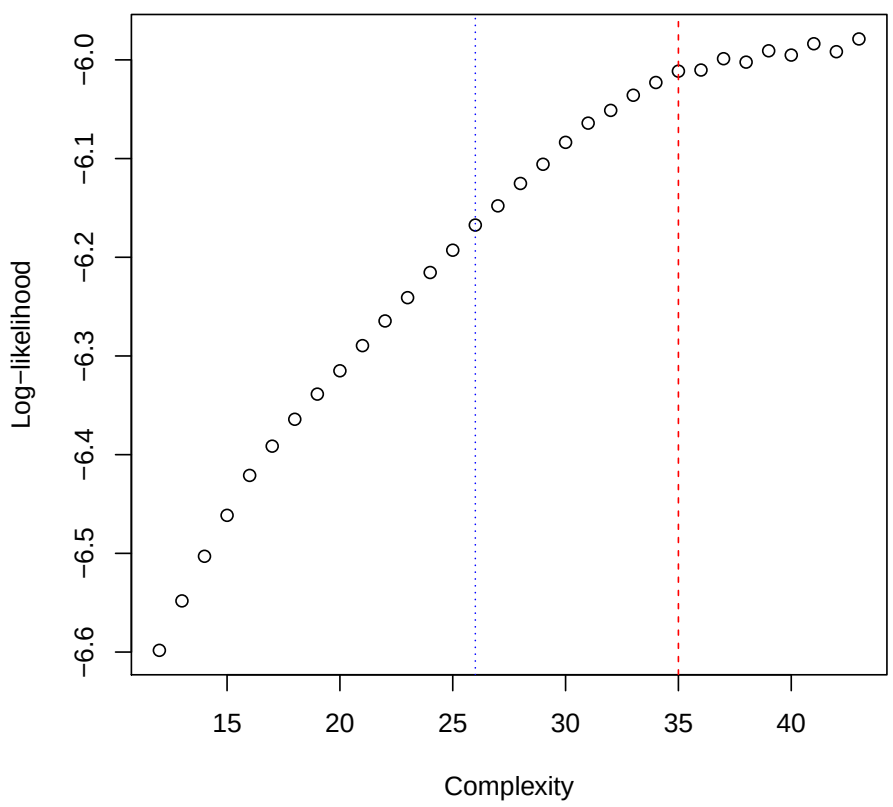
```
> set.seed(345)
> ## generate a 100-size sample from cnet6
> cnet6 <- cnRandomCatnet(numnodes=12, maxParents=5, numCategories=2)
> samples <- cnSamples(object=cnet6, numsamples=100, output="matrix")
> eval <- cnSearchOrder(data=samples, perturbations=NULL, maxParentSet=2, parentSizes=NULL, maxComplex
+         nodeOrder=order(runif(1:dim(samples)[1])), parentsPool=NULL, fixedParents=NULL,
+         echo=FALSE)
> ## now select a network based on AIC and plot it
> anet <- cnFindAIC(object=eval)
> anet

A catNetwork object with  12  nodes,  2  parents,  2  categories,
 Likelihood =  -6.011414 , Complexity =  35 .

> ## or BIC
> bnet <- cnFindBIC(object=eval, numsamples=dim(samples)[2])
> bnet

A catNetwork object with  12  nodes,  2  parents,  2  categories,
 Likelihood =  -6.167345 , Complexity =  26 .

> ## plot likelihood vs complexity for the resulting list of networks
> plot(eval@complexity, eval@loglik,
+ xlab="Complexity", ylab = "Log-likelihood",
+ main="Model selection: AIC and BIC complexities in red and blue.")
> abline(v=anet@complexity,lty=2,col="red")
> abline(v=bnet@complexity,lty=3,col="blue")
```



4.3 Stochastic Search in the Space of Orders

In cases when prior knowledge about the order of the nodes is not available, the user can employ a stochastic search method in the space of all possible node orders. In a way, this will break the general search problem into two smaller ones - 'order search' and 'search in order'. The space of orders includes all possible node permutations and, although still huge, is somehow smaller than that of all possible networks. This approach of order factorization is also implemented in [Friedman & Koller(2003)] but in a different, Bayesian, context. The idea of considering order learning as a sub-problem is also followed in [Larranaga et al.(1996)] and [Acid et al.(2001)] with constraint-based algorithms for conditional independence. In addition, *catnet* implements both frequentist, maximum likelihood based, and Bayesian learning frameworks.

The `catnet` package provides the stochastic search function `cnSearchSA`, which implements a Simulated Annealing (SA) algorithm, [Kirkpatrick et al.(1983)], [Cerny(1985)]. Simulated Annealing is a global optimization algorithm built on the Metropolis algorithm, [Metropolis et al.(1953)].

For a sample D_m of observations on variables $\mathcal{V}$ and an order Ω, we have defined $\hat{G}(\Omega, \alpha)$ to be the network with complexity α consistent with Ω that maximizes the log-likelihood 2. In fact, $\hat{G}$ may be only one representative network from a set of equivalent networks maximizing the likelihood. Moreover, we assume that Ω is a random element of the space of all permutations of nodes $\mathcal{V}$ and its probability is proportional to the likelihood of $\hat{G}(\Omega, \alpha)$

$$P(\Omega|D_m) \propto \exp(l(\hat{G}(\Omega, \alpha)|D_m)). \tag{3}$$

The functions `cnSearchSA` implements a Metropolis algorithm with acceptance probability

$$P(\Omega_2|\Omega_1,\beta) \propto 1_{\Omega_2 \in \mathcal{N}(\Omega_1)} \exp(-\min\{0, l(\hat{G}(\Omega_2,\alpha)|D_m) - l(\hat{G}(\Omega_1,\alpha)|D_m)\}/\beta), \tag{4}$$

where $\mathcal{N}(\Omega_1)$ is a neighborhood of Ω_1 and $\beta > 0$ is a parameter which specifies the temperature for the Simulating Annealing. The parameter β gradually decreases according to a cooling schedule specified by the function's parameters. The neighborhood $\mathcal{N}(\Omega)$ of an order Ω includes all orders obtained from Ω by repeating `orderShuffles` number of times the following exchange operation: a node index is randomly selected and exchanged with the next node index in Ω. By varying `orderShuffles`, the user can control the extent of $\mathcal{N}(\Omega)$.

The complexity α used in Eq. 4, also called target complexity, may vary from one iteration of SA to another. It is determined by a user specified selection criterion, the parameter `selectMode`. Note that the SA search is only optimal for the series of networks, one for each SA iteration, having complexities α.

In addition to the parameters of `cnSearchOrder`, `cnSearchSA` function also accepts

1. `selectMode` - determines how the target network complexity α is defined at each iteration of SA. It can be one of the 'AIC' and 'BIC' criteria. Any other value results in using `maxComplexity` for α.

2. `tempStart` - the starting temperature of the annealing process.

3. `tempCoolFact` - the cooling factor from one temperature step to another. It is a number between 0 and 1, inclusively. For example, if `tempStart` is the temperature in the first step, `tempStart*temthepCoolFact` will be the temperature in the second.

4. `tempCheckOrders` - the number of proposals to be checked for given temperature. This is the number of orders, elements of the current order neighborhood, to be proposed at each step before decreasing the temperature. Thus, if Ω is the current order at some temperature, totally `tempCheckOrders` orders from $\mathcal{N}(\Omega)$ will be proposed, and consequently accepted or rejected, before factoring down the temperature.

5. `maxIter` - the maximum number of orders to be checked. If for example `maxIter` is 40 and `tempCheckOrders` is 4, then 10 temperature decreasing steps will be eventually performed.

6. `orderShuffles` - a number that controls the extent of $\mathcal{N}(\Omega)$, the neighborhood of order Ω. The elements of $\mathcal{N}(\Omega)$ are obtained from Ω by `orderShuffles` number of switches of pairs of node indices.

7. `stopDiff` - a stopping criterion. If at a current temperature, after `tempCheckOrders` orders being checked, no likelihood improvement of level at least `stopDiff` is detected, then the SA stops and the function exists. Setting this parameter to zero guarantees exhausting all of the maximum allowed `maxIter` order searches.

8. `priorSearch` - a result from previous search in form of `catNetworkEvaluate` object. By setting this parameter, a new search can be initiated from the best order found in a previous search. Thus, a chain of searches can be constructed with varying SA control parameters providing greater adaptability and user control.

9. `numThreads` - the number of simultaneous threads run in parallel, each processing different order from the neighborhood of the current selected one.

The input set of parameters of `cnSearchSA` allows implementing a variety of search strategies, but selecting an optimal setup, naturally, depends on the data. We can only make the following suggestion to the user: try several parameter combinations, perform repeated search with limited number of

iterations (not too large `maxIter`) with each of them, choose a setting with the most consistent results (in terms of likelihood for a fixed complexity) and perform a longer search with the already chosen set of parameters. Another hint is to look at the acceptance rate of the SA algorithm and choose a setting that gives about 10 to 30 percent acceptance. In any case, a number of independent runs of `cnSearchSA` are recommended before making conclusions, a general recommendation for any MCMC procedure.

The following example theillustrates a typical call of `cnSearchSA`

```
> set.seed(345)
> samples <- cnSamples(object=cnet6, numsamples=100, output="matrix")
> netlist <- cnSearchSA(data=samples, perturbations=NULL,
+         maxParentSet=2, parentSizes=NULL, maxComplexity=20,
+         parentsPool=NULL, fixedParents=NULL,
+         tempStart=1, tempCoolFact = 0.9, tempCheckOrders = 4, maxIter = 40,
+         orderShuffles = 1, stopDiff = 0.0001,
+         priorSearch=NULL)
> bnet <- cnFind(netlist@nets, cnComplexity(cnet6))
> bnet

A catNetwork object with  12  nodes,  2  parents,  2  categories,
 Likelihood =  -6.628307 , Complexity =  20 .
```

As noted above, the function `cnSearchSA` has a parameter called `priorSearch`, which can take the result of previous call to `cnSearchSA`. In that case the new search starts where the previous search has ended thus trying to improve upon the best set of networks that have been already found. This mechanism allows implementation of sophisticated multi-stage search scenaria with more flexible user control.

By its nature, the search for best network according to a likelihood based score is NP-Complete, thus in general intractable, problem (see [Chickering(1996a)]). Inherently, the search functions implemented by *catnet* are highly intensive computationally and some means for processing in cluster environment are necessary.

The `cnSearchSA` function has a multi-threaded design. If k threads are started in parallel (`numThreads`=k), then the MC can run up to k times faster and cover correspondingly larger search space. It works as follows. From the neighborhood of the currently selected node order, k different candidate orders are chosen and exhaustive searches (equivalent to `cnSearchOrder`) for all of them are performed in parallel in one batch. After the batch is processed, based on the search results the acceptance/rejection decision for the candidate orders is taken in sequential manner. If one of these k searches succeeds to be accepted, according to the probability in Eq. 4, the corresponding node order is chosen as a current selection and the rest of the search results are discarded. Let $j \in [1, k]$ be the index of the first accepted order in the batch, then this batch processing step is equivalent to j steps of the baseline MC. On the other hand, if none of the k searches succeeds to be accepted, then, effectively, k steps of the MC are performed simultaneously, resulting in k fold speed up. When the acceptance rate of new node orders is low, as is usually the case, the performance boost is close to k fold.

4.4 Including Prior Probabilities

The estimation functionality of *catnet* can be further facilitated by including some prior node connectivity information. Both `cnSearchOrder` and `cnSearchSA` functions accepts the parameter `edgeProb`, which is a square matrix with numeric values in the range [0,1]. The $[i, j]$-th element in `edgeProb` specifies the probability of directed edge from j-th to i-th node. From a given edge probability matrix, a prior distribution on the space of networks is constructed based on the assumption that all edges are independent Bernoulli random variables. Formally, let G be a categorical Bayesian network and let the

binary variables $\delta_{ij} \in \{0, 1\}$ indicate the presence of edge from the j-th to the i-th node. Denote the elements of the edge probability matrix with q_{ij}. Then we define the prior probability of G as

$$\pi(G) \propto \prod_{i=1}^{n} \prod_{j \neq i} q_{ij}^{\delta_{ij}} (1 - q_{ij})^{1-\delta_{ij}}. \tag{5}$$

In the space of all possible networks, not necessarily DAGs, the above is a properly normalized distribution. Accordingly, we replace the objective log-likelihood function by the posterior score

$$\frac{1}{m} \sum_{i=1}^{n} \sum_{j=1}^{m} logP(X^j_{\Omega(i)} | \Pi_{\Omega(i)}(X^j)) + \sum_{i=1}^{n} \sum_{j \neq i} [\delta_{ij} log(q_{ij}) + (1 - \delta_{ij}) log(1 - q_{ij})], \tag{6}$$

where Ω is an order compatible with G.

The following example utilize the `edgeProb` parameter

```
> set.seed(678)
> numnodes <- 16
> numcats <- 3
> maxpars <- 2
> cnet8 <- cnRandomCatnet(numnodes, maxpars, numcats)
> ps <- cnSamples(cnet8, 500)
> ## next, a variable number of categories scanario is demonstrated
> ## find a node with descendants and reduce its number of categories
> mpars <- cnMatParents(cnet8)
> for(j in 1:numnodes)
+         if(sum(mpars[,j]) > 0)
+                 break
> if(j < numnodes)
+         cnet8@categories[[j]] <- cnet8@categories[[j]][1:(numcats-1)]
> ## now resets cnet8's probability table
> cnet8 <- cnSetProb(cnet8, ps)
> ## generate a new sample from the updated network
> ps <- cnSamples(cnet8, 500)
> res8 <- cnSearchOrder(data=ps, perturbations = NULL,
+                       maxParentSet = maxpars, parentSizes = NULL,
+                       maxComplexity = 0,
+                       nodeOrder = cnOrder(cnet8),
+                       parentsPool = NULL, fixedParents = NULL,
+                       edgeProb = NULL,
+                       echo = FALSE)
> anet8 <- cnFind(res8, cnComplexity(cnet8))
> cnCompare(cnet8, anet8)

 Edges:
                 TP = 6,
                 FP = 0,
                 FN = 0,
            F-score = 1.000000

> ## perform a stochastic search with a prior that favors the true edges (given by mpars)
> edgeHisto <- 0.5 + mpars / 4
> res9 <- cnSearchSA(data=ps, perturbations=NULL,
```

```
+                     maxParentSet=1, parentSizes = NULL,
+                     maxComplexity = 0,
+                     parentsPool = NULL, fixedParents = NULL, edgeProb = edgeHisto,
+                     selectMode = "BIC",
+                     tempStart = 1, tempCoolFact = 0.9, tempCheckOrders = 20,
+                     maxIter = 100, orderShuffles = -1, stopDiff = 1,
+                     numThreads = 2,
+                     priorSearch = NULL,
+                     echo=FALSE)
> anet9 <- cnFind(res9, cnComplexity(cnet8))
> cnCompare(cnet8, anet9)

Edges:
                 TP = 4,
                 FP = 4,
                 FN = 2,
            F-score = 0.794830,

Hamming:
            (FP+FN) = 6,
                exp = 9,

Skeleton:
                 TP = 5,
                 FP = 3,
                 FN = 1,

Order:
                 FP = 1,
                 FN = 1,

Markov blanket:
                 FP = 0,
                 FN = 1
```

The `dirProb` parameter, only available in the `cnSearchSA` function, specifies directional, also causal, prior information, thus allowing prior probabilities on the space of node orders to be defined. Similarly to `edgeProb`, `dirProb` is a square matrix R with numeric values r_{ij} in the range [0,1]. Its [i,j]-th element r_{ij} gives the probability of the j-th node to be before the i-th node. Since there are only two alternatives for each pair of nodes, the conditions $r_{ij} + r_{ji} = 1$ and $r_{ii} = 1$ are mandated. The prior probability of order Ω is then defined as

$$\pi(\Omega|R) \propto \Pi_{i<j} r_{ij}^{1(x_j \prec_\Omega x_i)} r_{ji}^{1(x_i \prec_\Omega x_j)}. \tag{7}$$

Accordingly, the posterior probability of Ω becomes

$$P(\Omega|D_m) \propto \exp(l(\hat{G}(\Omega, \alpha)|D_m))\pi(\Omega|R), \tag{8}$$

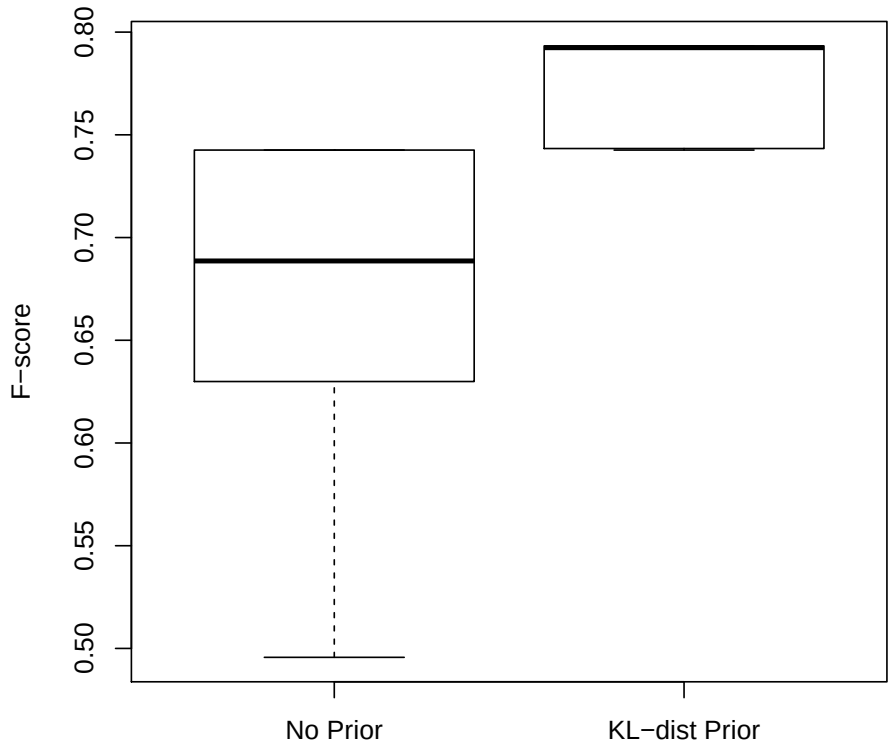
and the acceptance probability of the Metropolis algorithm changes to

$$P(\Omega_2|\Omega_1, \beta) \propto 1_{\Omega_2 \in \mathcal{N}(\Omega_1)} \exp(-\min\{0, h(\hat{G}(\Omega_2, \alpha)|D_m) - h(\hat{G}(\Omega_1, \alpha)|D_m)\}/\beta), \tag{9}$$

where $h(\hat{G}(\Omega, \alpha)|D_m) = l(\hat{G}(\Omega, \alpha)|D_m) + \log(\pi(\Omega|R))$.

The following example illustrates the usage of perturbations to estimate pairwise node directions and feed the `dirProb` parameter.

```
> cnSetSeed(3456)
> ncats <- 2
> cn <- cnRandomCatnet(20, 3, ncats, p.delta1=0.1, p.delta2=0.2)
> norder <- cnOrder(cn)
> numnodes <- cnNumNodes(cn)
> mpars <- cnMatParents(cn)
> numsamples <- 100
> ## simulate perturbations
> pert <- as.data.frame(matrix(rbinom(numnodes*numsamples, 1, p=0.25), ncol=numnodes))
> for(j in 1:numsamples)
+   for(i in 1:numnodes) {
+     if(pert[j,i])
+       pert[j,i] <- 1+floor(runif(1)*ncats)
+     for(ip in cn@parents[[i]]) {
+       if(pert[j,ip]) {
+         pert[j,i] <- 0
+       }
+     }
+   }
> ps <- cnSamples(cn, numsamples, pert, as.index=TRUE)
> ## pairwise conditional entropy difference between perturbed and non-perturbed data
> klmat <- cnEdgeDistanceKL(ps, pert)
> fscore1 <- NULL
> fscore2 <- NULL
> for(ntrials in 1:5) {
+   numiter <- 60
+   sares1 <- cnSearchSA(data=ps, perturbations=pert, maxParentSet=2,
+                        parentSizes=NULL, maxComplexity=0,
+                        parentsPool=NULL, fixedParents=NULL,
+                        edgeProb=NULL, dirProb=NULL, selectMode = "AIC",
+                        tempStart=0.1, tempCoolFact=0.9, tempCheckOrders=numiter,
+                        maxIter=numiter,
+                        orderShuffles=0, stopDiff=0,
+                        numThreads=2, priorSearch=NULL, echo=FALSE)
+   cmp <- cnCompare(cn, cnFind(sares1, cnComplexity(cn)))
+   fscore1 <- c(fscore1, cmp@fscore)
+   sares2 <- cnSearchSA(data=ps, perturbations=pert, maxParentSet=2,
+                        parentSizes=NULL, maxComplexity=0,
+                        parentsPool=NULL, fixedParents=NULL,
+                        edgeProb=NULL, dirProb=t(klmat), selectMode = "AIC",
+                        tempStart=0.1, tempCoolFact=0.9, tempCheckOrders=numiter,
+                        maxIter=numiter,
+                        orderShuffles=0, stopDiff=0,
+                        numThreads=2, priorSearch=NULL, echo=FALSE)
+   cmp <- cnCompare(cn, cnFind(sares2, cnComplexity(cn)))
+   fscore2 <- c(fscore2, cmp@fscore)
+ }
> pl <- list("No Prior"=fscore1, "KL-dist Prior"=fscore2)
> boxplot(pl, ylab="F-score")
```



5 Conclusion

Catnet package for `R` provides a framework for categorical Bayesian network inference. It implements a maximum likelihood based estimation with subsequent model selection of choice. Throughout support of data with missing values and perturbations is provided that makes the package suitable for microarray analysis. A Bayesian structure analysis is possible via hard-edge-constraints or prior distributions on node order and connectivity. A number of additional functions for sampling, prediction and network comparison complete the package to a self-contained tool-set for discrete network analysis.

References

[Acid et al.(2001)] Acid, S., Campos, L., Huete, J., The Search of Causal Orderings: A Short Cut for Learning Belief Networks. 2001, In Proc. 8-th conference on Uncertainty in Artificial Intelligence.

[Balov(2011)] Balov, N., Consistent Model Selection of Discrete Bayesian Networks from Incomplete Data 2011, submitted, arxiv: 1105.4507

[Beinlich et al.(1989)] Beinlich, I., Suermondth, G., Chavez, R., Cooper, G., The ALARM monitoring system. 1989, In Proc. 2-nd Euro. Conf. on AI and Medicine.

[Bouckaert(1992)] Bouckaert, R. Optimizing causal orderings for generating dags from data. 1992, In Proc. Conf. on Uncertainty in Artificial Intelligence, pages 9-16. Morgan-Kaufmann.

[Buntine(1996)] Buntine, W., A guide to the literature on learning probabilistic networks from data. 1996, IEEE Transaction on Knowledge and Data Engineering, 8:195-210.

[Cerny(1985)] Cerny, V., A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm, 1985, Journal of Optimization Theory and Applications, 45:41-51

[Chickering(1996a)] Chickering, D. M., Learning Bayesian Networks is NP-Complete. 1996(a), In D. Fisher, H. Lenz (Eds.) Learning from Data: Artificial Intelligence and Statistics V, Ch. 12, 121-130.

[Chickering(1996b)] Chickering, D. M., Learning Equivalence Classes of Bayesian-Network Structures. 1996(b), Journal of Machine Learning research, 2, pp. 445-498.

[Cooper & Herskovitz(1992)] Cooper, G. F., Herskovitz, E., A Bayesian method for the induction of probabilistic networks from data. 1992, Mach. Learn., 9, pp. 309-347

[Daly & Shen(2007)] Daly, R., Shen, Q., Methods to Accelerate the Learning of Bayesian Network Structures. 2007, In Proc. of the UK Workshop on Computational Intelligence, Imperial College, London.

[Heckeman et al.(1995)] Heckeman, D., Geiger, D., Chickering, D., Learning Bayesian networks: The combination of knowledge and statistical data. 1995, Machine Learning, 20(3), pp. 197-243.

[Gentleman et al.(2009)] Gentleman, R., Whalen, E., Huber, W., Falcon, S., graph: A package to handle graph data structures. R package. 2009, package version 1.24.1

[graphvis] Graphviz - Graph Visualization Software http://www.graphviz.org/

[Friedman et al.(1999)] Friedman, N., Goldszmidt, M., Wyner, A., Data Analysis with Bayesian Networks: A Bootstrap Approach. 1999, Proc. Fifteenth Conf. on Uncertainty in Artificial Intelligence (UAI).

[Friedman et al.(2000)] Friedman, N., Goldszmidt, M., Wyner, A., Using Bayesian Networks to Analyze Expression Data. 2000, Journal of Computational Biology, 7, pp. 601-620.

[Friedman & Koller(2003)] Friedman, N., Koller, D., Being Bayesian about network structure: A Bayesian approach to structure discovery in Bayesian networks. 2003, Mach. Learn., 50, pp. 95-125.

[Kirkpatrick et al.(1983)] Kirkpatrick, S.; C. D. Gelatt, M. P. Vecchi., Optimization by Simulated Annealing, 1983, Science. New Series 220.

[Larranaga et al.(1996)] Larranaga, P. , Kuijpers, C., Murga, R., Yurramendi, Y., Learning Bayesian network structures by searching for the best ordering with genetic algorithms. 1996, IEEE Trans. Pattern Anal. and Mach. Intell.,26:487-493.

[Metropolis et al.(1953)] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N. , Teller, A.H. and Teller, E., Equations of State Calculations by Fast Computing Machines, 1953, Journal of Chemical Physics, 21(6):1087-1092.

[Neapolitan(2003)] Neapolitan, R., E., Learning Bayesian Networks. 2003, Prentice Hall.

[Pearl & Verma(1988)] Pearl, J., Verma,T.S., A theory of inferred causation. 1991, 2-nd Conference on the Principles of Knowledge Representation and Reasoning, Cambridge, MA.

[Pearl(1988)] Pearl, J., Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. 1988, Morgan Kaufmann.

[Salzman & Almudevar(2006)] Salzman, P., Almudevar, A., Using Complexity for the Estimation of Bayesian Networks. 2006, Statistical Applications in Genetics and Molecular Biology, Vol. 5, Issue 1.

[Scutari(2010)] Scutari, M., bnlearn: Bayesian network structure learning. R package. 2010, package version 1.8

[Sebastiani et al.(2004)] Sebastiani, P., Abad, M., Ramoni, M., Bayesian Networks for Genomic Analysis. 2004, Genomic Signal Processing and Statistics, pp. 281-320.

[Tierney et al.(2008)] Tierney, L., Rossini, A. J., Na Li, Sevcikova, H., snow: Simple Network of Workstations. R package. 2008, package version 0.3-3

[Tsamardinos et al.(2003)] Tsamardinos, I., Aliferis, C., Statnikov, A., Algorithms for Large Scale Markov Blanket Discovery. 2003, In Proc. of the 16-th Inter. Florida Artificial Intelligence Research Society Conference, pp. 376-381, AAAI Press.

[Verma & Pearl(1990)] Verma, T., Pearl, J., Equivalence and synthesis of causal models. 1990, In Henrion,M., Shachter, R., Kanal, L., and Lemmer, J., editors, Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence, pp. 220-227.

[Yaramakala & Margaritis(2005)] Yaramakala, S., Margaritis, D., Speculative Markov Blanket Discovery for Optimal Feature Selection. 2005, In ICDM'05, Proceedings of the 5-th IEEE Conference on Data Mining, pp. 809-812.