Learning Bayesian Networks in R

an Example in Systems Biology

Marco Scutari

m.scutari@ucl.ac.uk
Genetics Institute
University College London

July 9, 2013

Bayesian Networks Essentials

Bayesian Networks

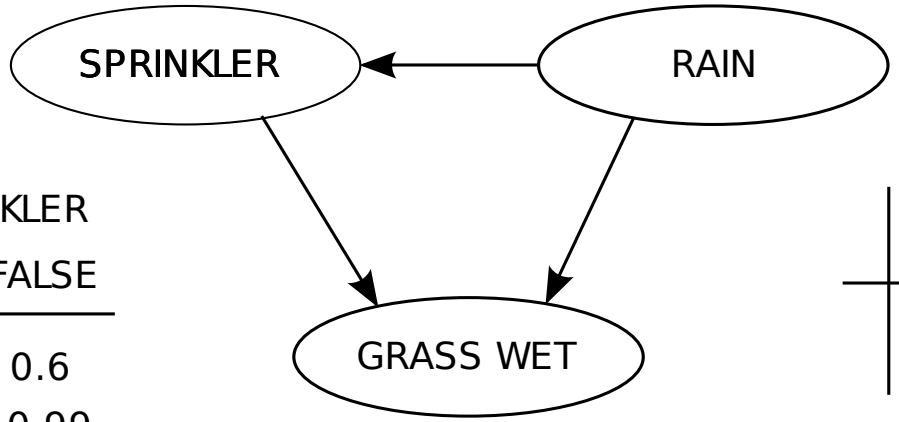
Bayesian networks [21, 27] are defined by:

- a network structure, a directed acyclic graph $\mathcal{G} = (\mathbf{V}, A)$, in which each node $v_i \in \mathbf{V}$ corresponds to a random variable X_i;
- a global probability distribution, $\mathbf{X}$, which can be factorised into smaller local probability distributions according to the arcs $a_{ij} \in A$ present in the graph.

The main role of the network structure is to express the conditional independence relationships among the variables in the model through graphical separation, thus specifying the factorisation of the global distribution:

$$\mathrm{P}(\mathbf{X}) = \prod_{i=1}^{p} \mathrm{P}(X_i \mid \Pi_{X_i}) \quad \text{where} \quad \Pi_{X_i} = \{\text{parents of } X_i\}$$

A Simple Bayesian Network: Watson's Lawn



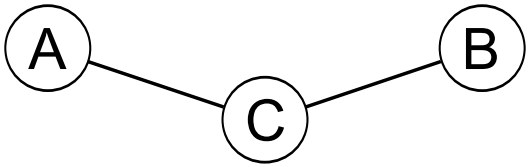
RAIN	SPRINKLER TRUE	SPRINKLER FALSE
FALSE	0.4	0.6
TRUE	0.01	0.99

RAIN TRUE	RAIN FALSE
0.2	0.8

SPRINKLER	RAIN	GRASS WET TRUE	GRASS WET FALSE
FALSE	FALSE	0.0	1.0
FALSE	TRUE	0.8	0.2
TRUE	FALSE	0.9	0.1
TRUE	TRUE	0.99	0.01

Graphical Separation

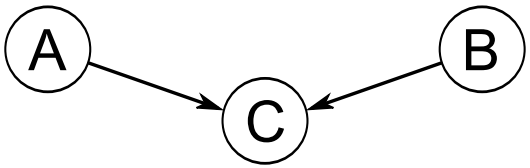
separation (undirected graphs)



$$\mathbf{A} \perp\!\!\!\perp \mathbf{B} \,|\, \mathbf{C}$$

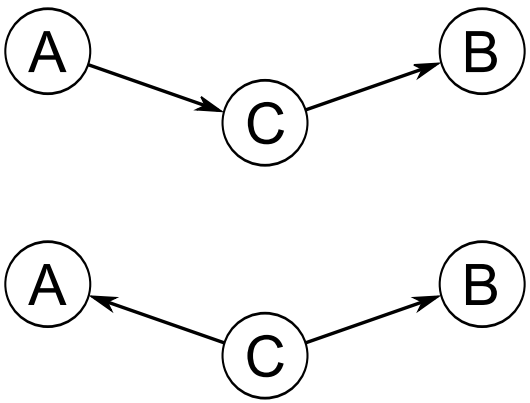
$$\mathrm{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \mathrm{P}(\mathbf{A} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{B} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{C})$$

d-separation (directed acyclic graphs)



$$\mathbf{A} \not\perp\!\!\!\perp \mathbf{B} \,|\, \mathbf{C}$$

$$\mathrm{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \mathrm{P}(\mathbf{C} \,|\, \mathbf{A}, \mathbf{B})\, \mathrm{P}(\mathbf{A})\, \mathrm{P}(\mathbf{B})$$



$$\mathbf{A} \perp\!\!\!\perp \mathbf{B} \,|\, \mathbf{C}$$

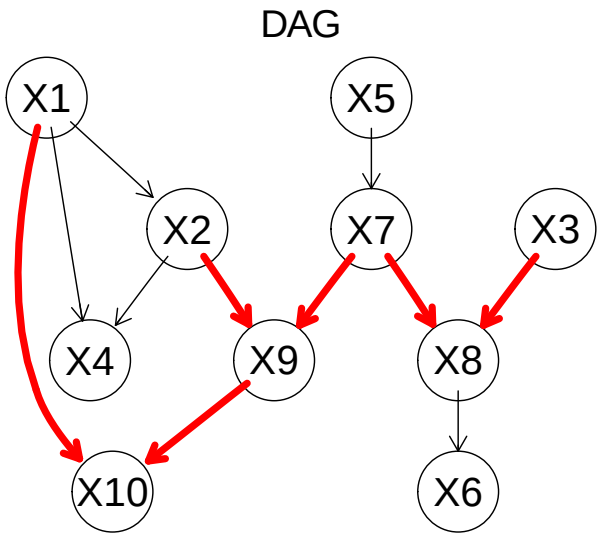
$$\begin{aligned} \mathrm{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}) &= \\ &= \mathrm{P}(\mathbf{B} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{C} \,|\, \mathbf{A})\, \mathrm{P}(\mathbf{A}) \\ &= \mathrm{P}(\mathbf{A} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{B} \,|\, \mathbf{C})\, \mathrm{P}(\mathbf{C}) \end{aligned}$$

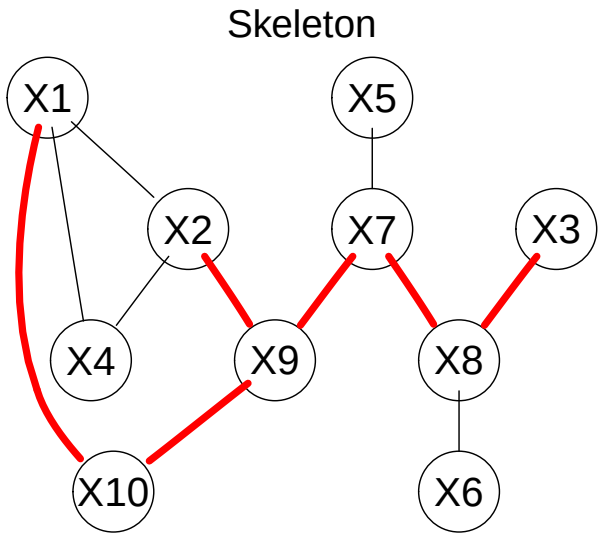
Skeletons, Equivalence Classes and Markov Blankets

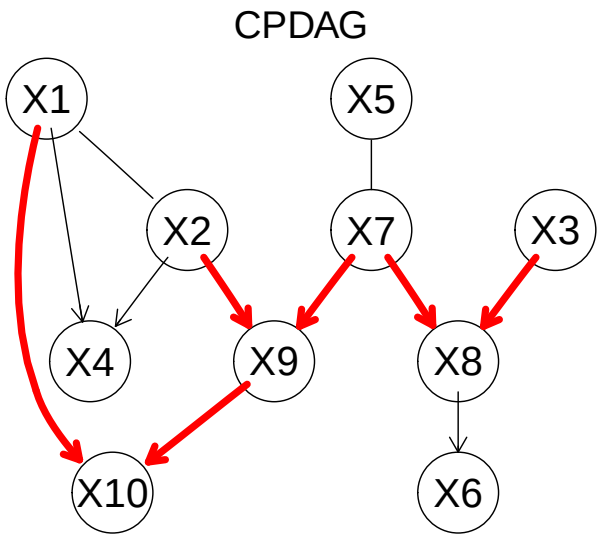
Some useful quantities in Bayesian network modelling:

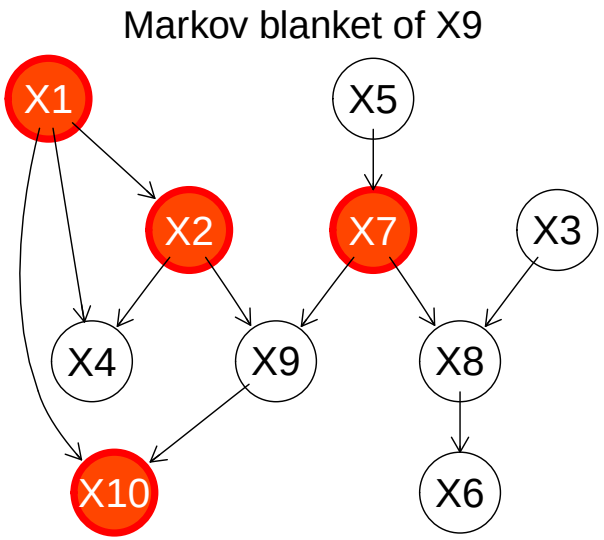
- The skeleton: the undirected graph underlying a Bayesian network, i.e. the graph we get if we disregard arcs' directions.
- The equivalence class: the graph (CPDAG) in which only arcs that are part of a v-structure (i.e. $A \rightarrow C \leftarrow B$) and/or might result in a v-structure or a cycle are directed. All valid combinations of the other arcs' directions result in networks representing the same dependence structure P.
- The Markov blanket of a node X_i, the set of nodes that completely separates X_i from the rest of the graph. Generally speaking, it is the set of nodes that includes all the knowledge needed to do inference on X_i, from estimation to hypothesis testing to prediction: the parents of X_i, the children of X_i, and those children's other parents.

Skeletons, Equivalence Classes and Markov Blankets









Learning a Bayesian Network

Model selection and estimation are collectively known as learning, and are usually performed as a two-step process:

1. structure learning, learning the network structure from the data;
2. parameter learning, learning the local distributions implied by the structure learned in the previous step.

This workflow is implicitly Bayesian; given a data set $\mathcal{D}$ and if we denote the parameters of the global distribution as $\mathbf{X}$ with Θ, we have

$$\underbrace{\mathrm{P}(\mathcal{M} \mid \mathcal{D}) = \mathrm{P}(\mathcal{G}, \Theta \mid \mathcal{D})}_{learning} = \underbrace{\mathrm{P}(\mathcal{G} \mid \mathcal{D})}_{structure\,learning} \cdot \underbrace{\mathrm{P}(\Theta \mid \mathcal{G}, \mathcal{D})}_{parameter\,learning}$$

and structure learning is done in practice as

$$\mathrm{P}(\mathcal{G} \mid \mathcal{D}) \propto \mathrm{P}(\mathcal{G})\,\mathrm{P}(\mathcal{D} \mid \mathcal{G}) = \mathrm{P}(\mathcal{G}) \int \mathrm{P}(\mathcal{D} \mid \mathcal{G}, \Theta)\,\mathrm{P}(\Theta \mid \mathcal{G}) d\Theta.$$

Inference on Bayesian Networks

Inference on Bayesian networks usually consists of conditional probability (CPQ) or maximum a posteriori (MAP) queries.

Conditional probability queries are concerned with the distribution of a subset of variables $\mathbf{Q} = \{X_{j_1}, \ldots, X_{j_l}\}$ given some evidence $\mathbf{E}$ on another set $X_{i_1}, \ldots, X_{i_k}$ of variables in $\mathbf{X}$:

$$CPQ(\mathbf{Q} \mid \mathbf{E}, \mathcal{M}) = \mathrm{P}(\mathbf{Q} \mid \mathbf{E}, \mathcal{G}, \Theta) = \mathrm{P}(X_{j_1}, \ldots, X_{j_l} \mid \mathbf{E}, \mathcal{G}, \Theta).$$

Maximum a posteriori queries are concerned with finding the configuration $\mathbf{q}^*$ of the variables in $\mathbf{Q}$ that has the highest posterior probability:

$$MAP(\mathbf{Q} \mid \mathbf{E}, \mathcal{M}) = \mathbf{q}^* = \operatorname*{argmax}_{\mathbf{q}} \mathrm{P}(\mathbf{Q} = \mathbf{q} \mid \mathbf{E}, \mathcal{G}, \Theta).$$

Causal Protein-Signalling Network from Sachs et al.

Source

What follows reproduces (to the best of my ability, and Karen Sachs' recollections about the implementation details that did not end up in the Methods section) the statistical analysis in the following paper [29] from my book [25]:

Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data
Karen Sachs, *et al.*
Science **308**, 523 (2005);
DOI: 10.1126/science.1105809

That's a landmark paper in applying Bayesian Networks because:

- it highlights the use of observational vs interventional data;
- results are validated using existing literature.

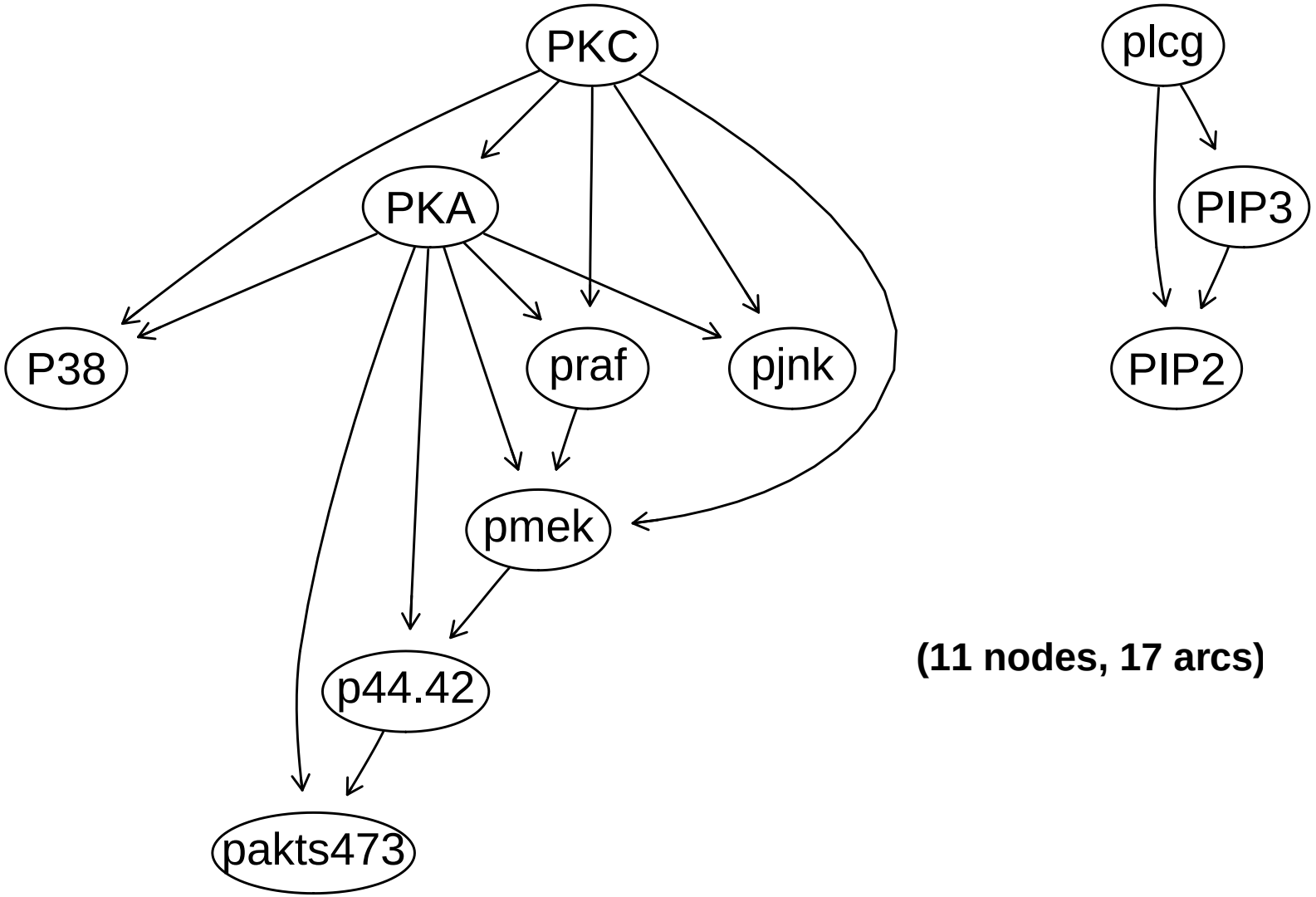
An Overview of the Data

The data consist in the simultaneous measurements of 11 phosphorylated proteins and phospholypids derived from thousands of individual primary immune system cells:

- 1800 data subject only to general stimolatory cues, so that the protein signalling paths are active;
- 600 data with with specific stimolatory/inhibitory cues for each of the following 4 proteins: pmek, PIP2, pakts473, PKA;
- 1200 data with specific cues for PKA.

Overall, the data set contains 5400 observations with no missing value.

Network Validated from Literature



(11 nodes, 17 arcs)

Plotting the Network

The plot in the previous slide requires **bnlearn** [25] and **Rgraphviz** [14] (which is based on **graph** [13] and the Graphviz library).

```
> library(bnlearn)
> library(Rgraphviz)
> spec =
+   paste("[PKC][PKA|PKC][praf|PKC:PKA][pmek|PKC:PKA:praf]",
+         "[p44.42|pmek:PKA][pakts473|p44.42:PKA][P38|PKC:PKA]",
+         "[pjnk|PKC:PKA][plcg][PIP3|plcg][PIP2|plcg:PIP3]")
> net = model2network(spec)
> class(net)
[1] "bn"
> graphviz.plot(net, shape = "ellipse")
```

The spec string specifies the structure of the Bayesian network in a format that recalls the decomposition into local probabilities; the order of the variables is irrelevant.

Advanced Plotting: Highlighting Arcs and Nodes

`graphviz.plot()` is simpler to use (but less flexible) than the functions in **Rgraphviz**; we can only choose the `layout` and do some limited formatting using `shape` and `highlight`.

```
> h.nodes = c("praf", "pmek", "p44.42", "pakts473")
> high = list(nodes = h.nodes, arcs = arcs(subgraph(net, h.nodes)),
+ col = "darkred", fill = "orangered", lwd = 2, textCol = "white")
> gr = graphviz.plot(net, shape = "ellipse", highlight = high)
```

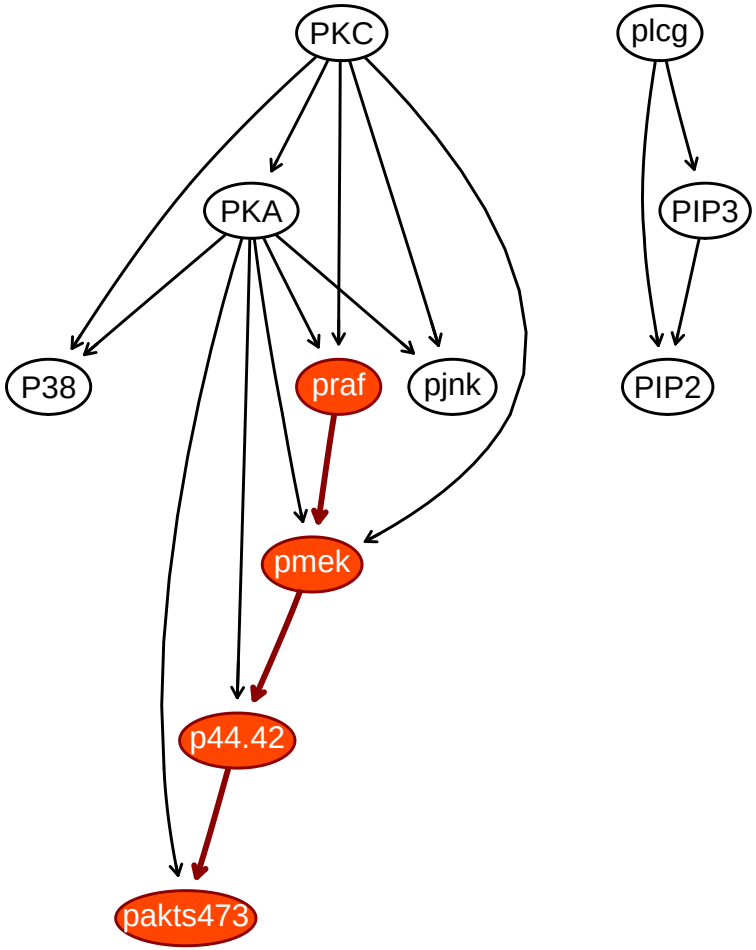
`graphviz.plot()` returns a `graphNEL` object, which can be customised with the functions in **graph** and **Rgraphviz**.

```
> nodeRenderInfo(gr)$col[c("PKA", "PKC")] = "darkgreen"
> nodeRenderInfo(gr)$fill[c("PKA", "PKC")] = "limegreen"
> edgeRenderInfo(gr)$col[c("PKA~praf", "PKC~praf")] = "darkgreen"
> edgeRenderInfo(gr)$lwd[c("PKA~praf", "PKC~praf")] = 2
> renderGraph(gr)
```

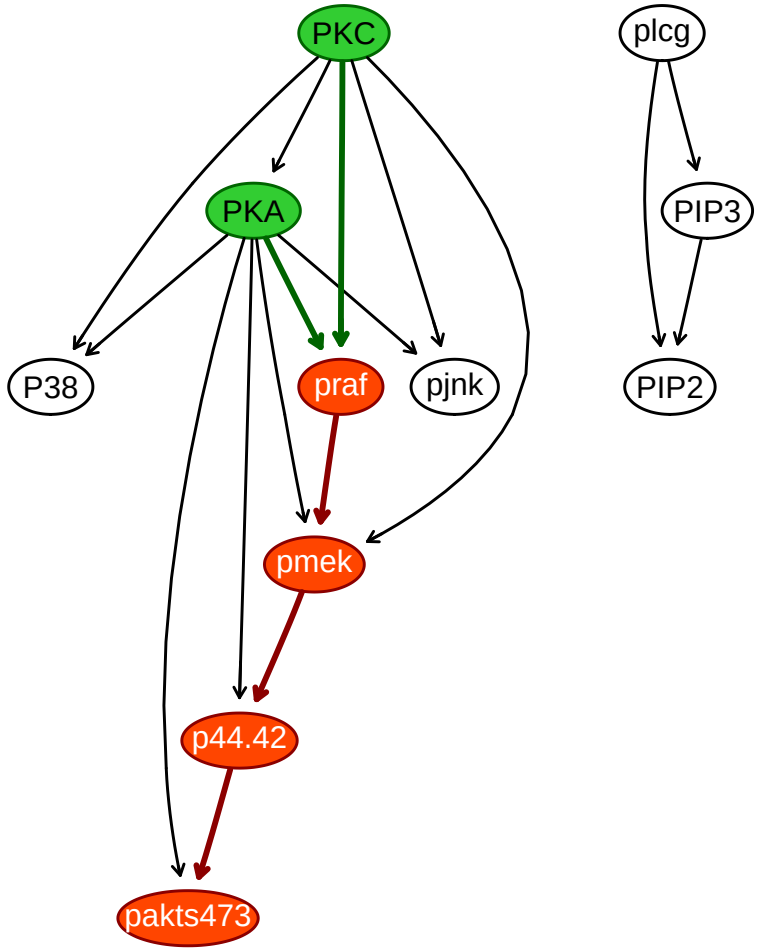
To achieve a complete control on the layout of the network, we can export gR to the **igraph** [6] package or use **Rgraphviz** directly.

Plotting Networks, with Formatting

graphviz.plot(...)



renderGraph(...)



Creating a Network Structure in **bnlearn**

- With the network's string representation, using `model2network()` and `modelstring()`.

```
> model2network(modelstring(net))
```

- Creating an empty network and adding arcs one at a time.

```
> e = empty.graph(nodes(net))
> e = set.arc(e, from = "PKC", to = "PKA")
```

- Creating an empty network and adding all arcs in one batch.

```
> to.add = matrix(c("PKC", "PKA", "praf", "PKC"), ncol = 2,
+                 byrow = TRUE, dimnames = list(NULL, c("from", "to")))
> to.add
```

```
     from   to
[1,] "PKC"  "PKA"
[2,] "praf" "PKC"
```

```
> arcs(e) = to.add
```

Creating a Network Structure in **bnlearn**

- Creating an empty network and adding all arcs using an adjacency matrix.

```
> n.nodes = length(nodes(e))
> adj = matrix(0, nrow = n.nodes, ncol = n.nodes)
> colnames(adj) = rownames(adj) = nodes(e)
> adj["PKC", "PKA"] = 1
> adj["praf", "PKC"] = 1
> adj["praf", "PKA"] = 1
> amat(e) = adj
> bnlearn:::fcat(modelstring(e))
```

```
[P38][p44.42][pakts473][PIP2][PIP3][pjnk][plcg][pmek][praf]
[PKC|praf][PKA|PKC:praf]
```

- Creating one or more random networks.

```
> random.graph(nodes(net), num = 5, method = "melancon")
```

Gaussian Bayesian Networks

Using Only Observational Data

As a first, exploratory analysis, we can try to learn a network from the data that were subject only to general stimolatory cues. Since these cues only ensure the pathways are active, but do not tamper with them in any way, such data are observational (as opposed to interventional).

```
> sachs = read.table("sachs.data.txt", header = TRUE)
> head(sachs, n = 4)
  praf pmek  plcg PIP2  PIP3 p44.42 pakts473 PKA   PKC  P38 pjnk
1 26.4 13.2  8.82 18.3 58.80   6.61     17.0 414 17.00 44.9 40.0
2 35.9 16.5 12.30 16.8  8.13  18.60     32.5 352  3.37 16.5 61.5
3 59.4 44.1 14.60 10.2 13.00  14.90     32.5 403 11.40 31.9 19.5
4 73.0 82.8 23.10 13.5  1.29   5.83     11.8 528 13.70 28.6 23.1
```

Most approaches in literature cannot handle interventional data, but work "out of the box" with observational ones.

Gaussian Bayesian Networks

When dealing with continuous data, we often assume they follow a multivariate normal distribution to fit a Gaussian Bayesian network [12, 26]. The local distribution of each node is a linear model,

$$X_i \quad = \mu + \Pi_{X_i}\boldsymbol{\beta} + \varepsilon \quad \text{with} \quad \varepsilon \sim N(0, \sigma_i).$$

which can be estimated with any frequentist or Bayesian approach. The same holds for the network structure:

- Constraint-based algorithms [24, 31, 32] use statistical tests to learn conditional independence relationships from the data.
- In score-based algorithms [23, 28], each candidate network is assigned a goodness-of-fit score, which we want to maximise.
- Hybrid algorithms [11, 33] use conditional independence tests are to restrict the search space for a subsequent score-based search.

Structure Learning: Constraint-Based Algorithms

```
> library(bnlearn)
> print(iamb(sachs))
  Bayesian network learned via Constraint-based methods
  model:
    [partially directed graph]
  nodes:                                  11
  arcs:                                   8
    undirected arcs:                      6
    directed arcs:                        2
  average markov blanket size:            1.64
  average neighbourhood size:             1.45
  average branching factor:               0.18

  learning algorithm:                     Incremental Association
  conditional independence test:          Pearson's Linear Correlation
  alpha threshold:                        0.05
  tests used in the learning procedure:   259
  optimized:                              TRUE
```

Structure Learning: Score-Based Algorithms

```
> print(hc(sachs))
  Bayesian network learned via Score-based methods
  model:
   [praf][PIP2][p44.42][PKC][pmek|praf][PIP3|PIP2][pakts473|p44.42]
   [P38|PKC][plcg|PIP3][PKA|p44.42:pakts473][pjnk|PKC:P38]
  nodes:                                 11
  arcs:                                  9
    undirected arcs:                     0
    directed arcs:                       9
  average markov blanket size:           1.64
  average neighbourhood size:            1.64
  average branching factor:              0.82

  learning algorithm:                    Hill-Climbing
  score:                 Bayesian Information Criterion (Gaussian)
  penalization coefficient:              3.37438
  tests used in the learning procedure:  145
  optimized:                             TRUE
```

Structure Learning: Hybrid Algorithms

```
> print(mmhc(sachs))
  Bayesian network learned via Hybrid methods
  model:
   [praf][PIP2][p44.42][PKC][pmek|praf][PIP3|PIP2][pakts473|p44.42]
   [P38|PKC][pjnk|PKC][plcg|PIP3][PKA|p44.42:pakts473]
  nodes:                                    11
  arcs:                                     8
  [...]

  learning algorithm:                       Max-Min Hill-Climbing
  constraint-based method:                  Max-Min Parent Children
  conditional independence test:            Pearson's Linear Correlation
  score-based method:                       Hill-Climbing
  score:                    Bayesian Information Criterion (Gaussian)
  alpha threshold:                          0.05
  penalization coefficient:                 3.37438
  tests used in the learning procedure:     106
  optimized:                                TRUE
```

Structure Learning: Additional Arguments

Since defaults are (often) not appropriate, we can tune each structure learning algorithm in **bnlearn** with several optional arguments.

- Constraint-based algorithms: we can pick the test [8, 26], the alpha threshold and the number of permutations, e.g.:

```
> inter.iamb(sachs, test = "smc-cor", B = 100, alpha = 0.01)
```

- Score-based algorithms: we can pick the score function [17, 12] and its tuning parameters, the number of random restarts, the length of the tabu list, the maximum number of iterations, and more, e.g.:

```
> hc(sachs, score = "bge", iss = 3, restart = 5, perturb = 10)
> tabu(sachs, tabu = 15, max.iter = 500)
```

- Hybrid algorithms: both the above, e.g.:

```
> rsmax2(sachs, restrict = "si.hiton.pc", maximize = "tabu",
+   test = "zf", alpha = 0.01, score = "bic-g")
```

Other useful arguments: `debug`, `whitelist`, `blacklist`.

Parameter Learning: Fitting and Modifying

```
> net = hc(sachs)
> bn = bn.fit(net, sachs, method = "mle")
> bn$pmek

  Parameters of node pmek (Gaussian distribution)
Conditional density: pmek | praf
Coefficients:
(Intercept)         praf
  -0.834129     0.520336
Standard deviation of the residuals: 16.72394

> bn$pmek = list(coef = c(0, 0.5), sd = 20)
> bn$pmek

  Parameters of node pmek (Gaussian distribution)
Conditional density: pmek | praf
Coefficients:
(Intercept)         praf
        0.0          0.5
Standard deviation of the residuals: 20
```

Parameter Learning: with Undirected Arcs

When we learn a CPDAG representing an equivalence class (e.g. with constraint-based algorithms), such as

```
> pdag = iamb(sachs)
```

we must set the directions of the undirected arcs before learning the parameters. We can do that automatically with `cextend` [7]

```
> dag = cextend(pdag)
```

by imposing a topological ordering on the nodes,

```
> dag = pdag2dag(pdag, ordering = node.ordering(net))
```

or by hand for each arc.

```
> pdag = set.arc(pdag, from = "praf", to = "pmek",
+           check.cycles = FALSE)
```

Parameter Learning: Other Methods

We can use a list containing `coef`, `sd`, `fitted` and `resid` to set each node parameters' from other models, either replacing parts of a Bayesian network returned by `bn.fit()` or creating a new one with `custom.fit()`.

```
> dPKA = list(coef = c("(Intercept)" = 1, "PKC" = 1), sd = 2))
> dPKC = list(coef = c("(Intercept)" = 1), sd = 2))
> bn = custom.fit(net, dist = list(PKA = dPKA, PKC = dPKC, ...))
```

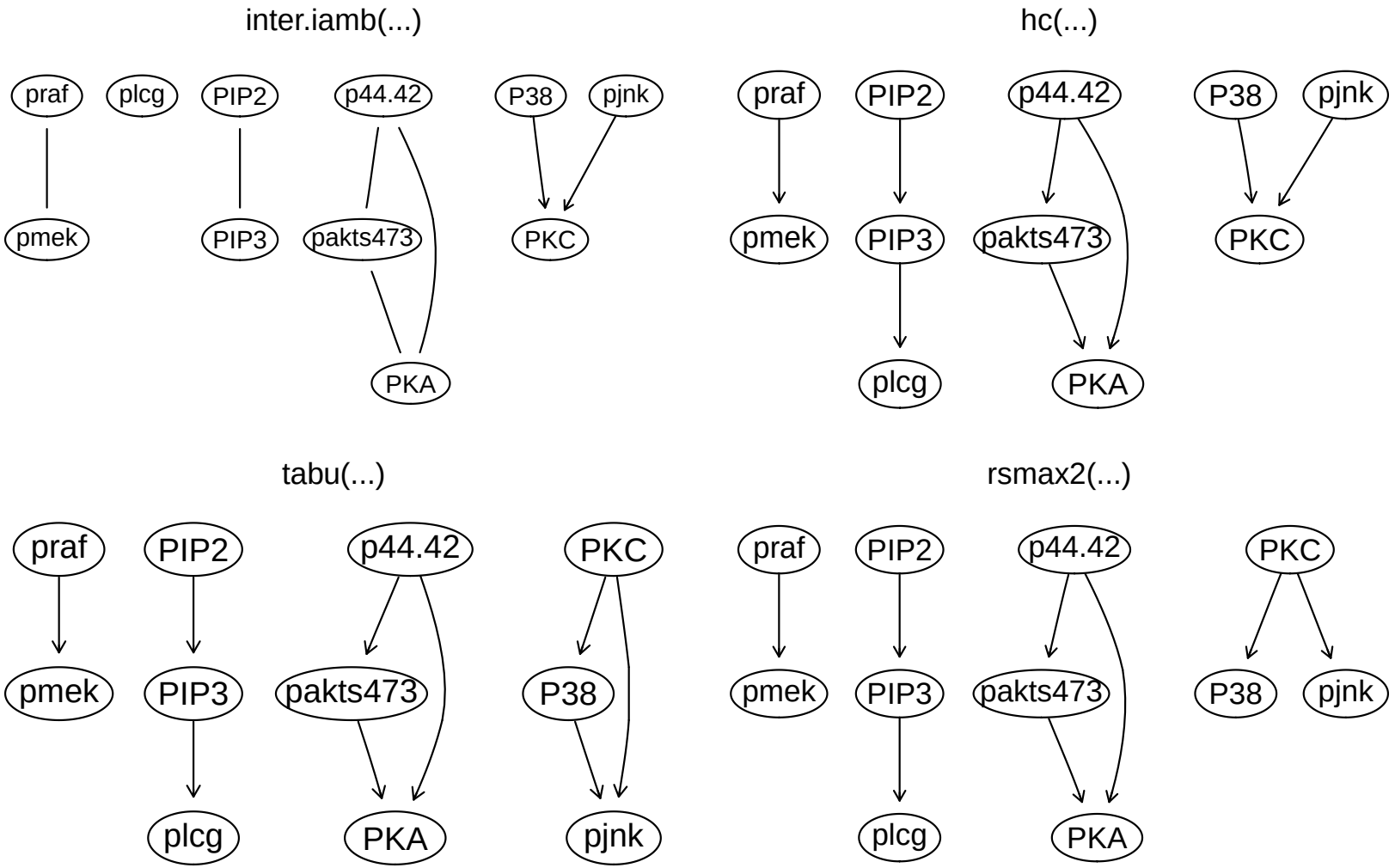
There are shortcuts to do that directly for the **penalized** package [15],

```
> library(penalized)
> bn$pmek = penalized(pmek, penalized = ~ praf, data = sachs,
+                lambda2 = 0.1, trace = FALSE)
```

and for `lm()` and `glm()`:

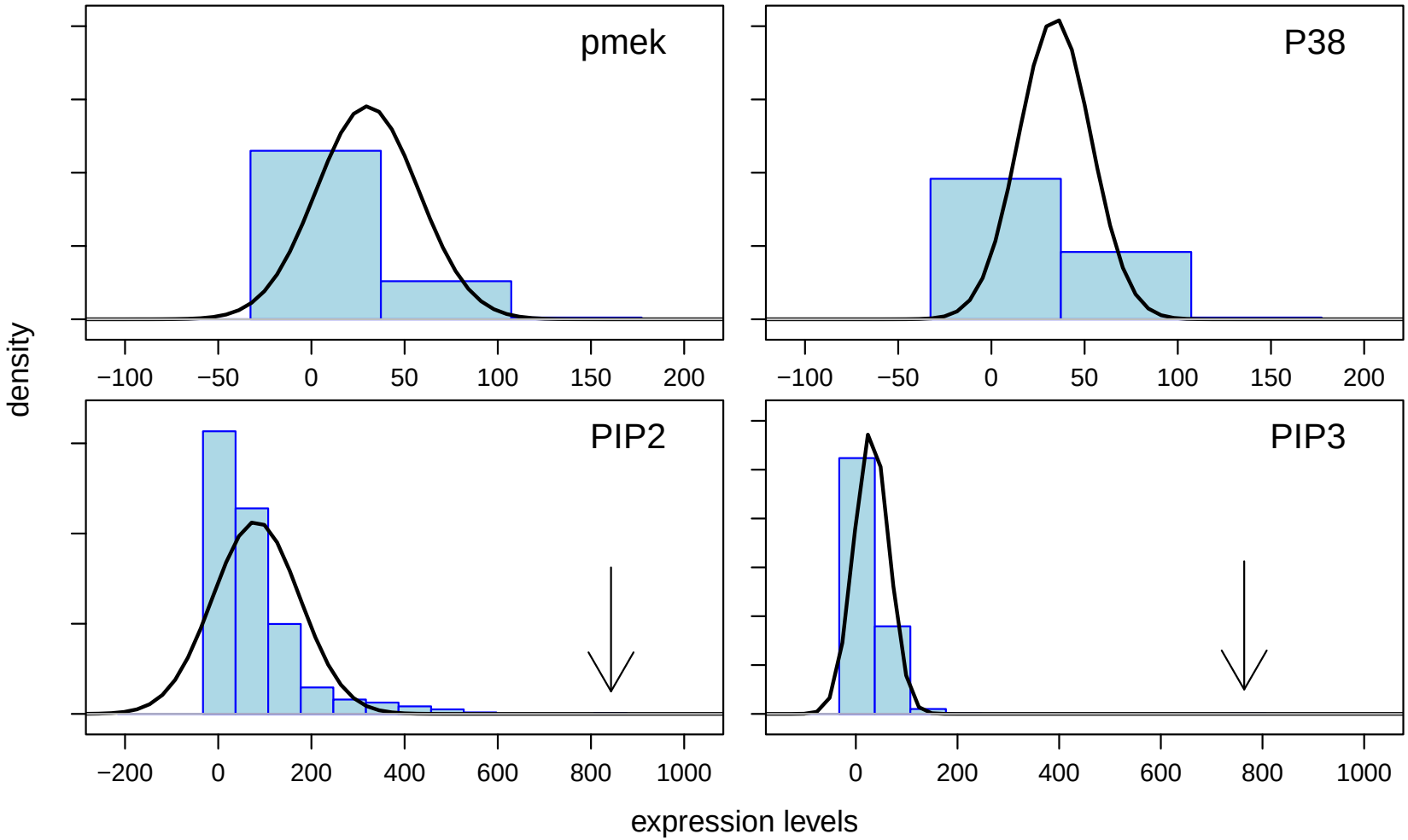
```
> bn.pmek = lm(pmek ~ praf, data = sachs, na.action = "na.exclude",
+                weight = runif(nrow(sachs)))
```

What Kind of Network Structures Did We Learn?



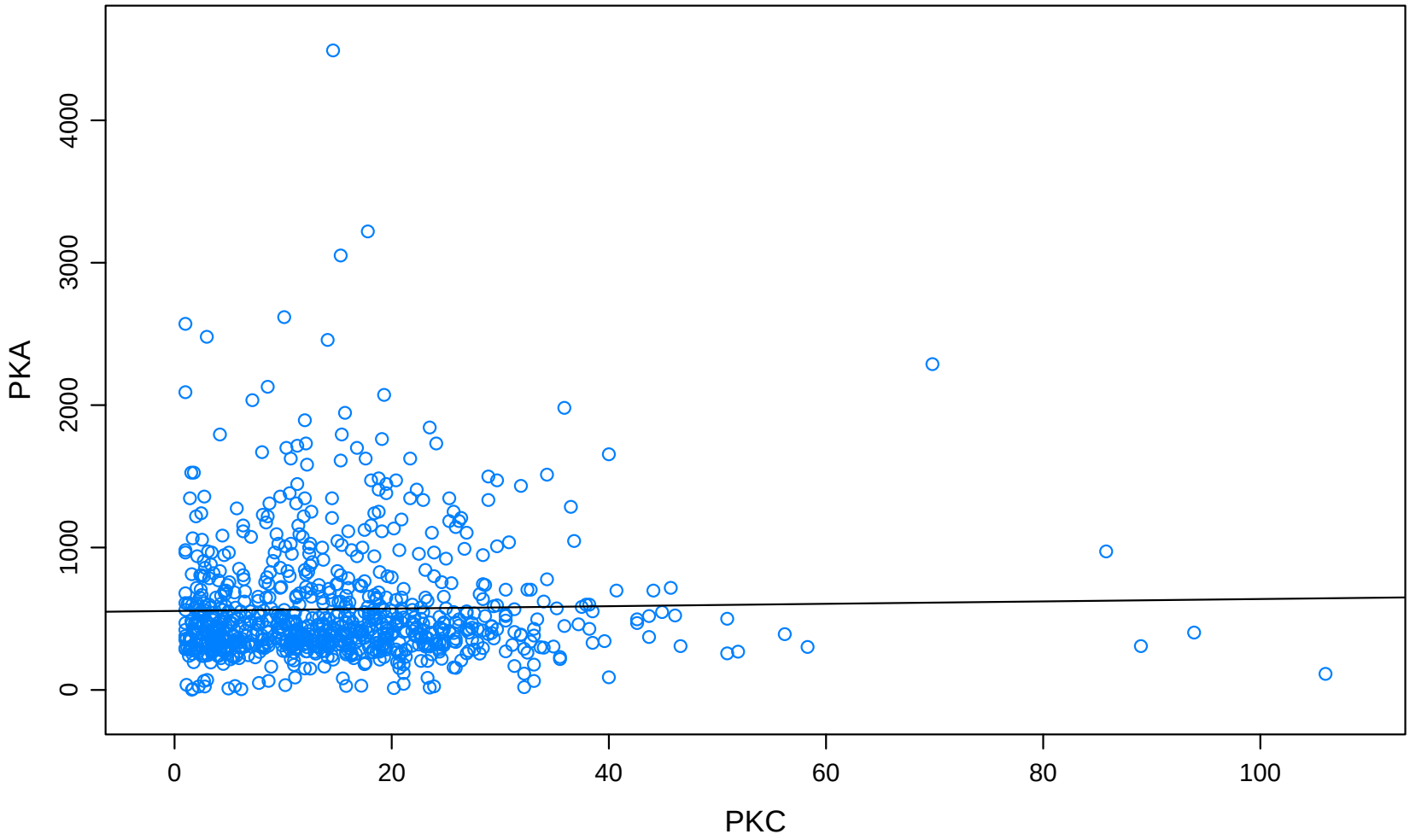
They look nothing like the one validated from literature...

Parametric Assumptions: Variables Are Not Normal



They are not even symmetric...

Parametric Assumptions: Dependencies Are Not Linear



The regression line is nearly flat...

Discrete Bayesian Networks

Transforming the Data & Parametric Assumptions

Since we cannot use Gaussian Bayesian networks on the raw data, we must transform them and possibly change our parametric assumptions. For example, we can:

- transform each variable using a Box-Cox transform [35] to make it normal (or at least symmetric);
- discretise each variable [20, 34] using quantiles or fixed-length intervals;
- jointly discretise all the variables [16] into a small number of intervals by iteratively collapsing the intervals defined by their quantiles.

The last choice is the only one that gets rid of both normality and linearity assumptions while trying to preserve the dependence structure of the data.

Discretize with Hartemink's Method

Hartemink's method [16] is designed to preserve pairwise dependencies as much as possible, unlike marginal discretisation methods.

```
> library(bnlearn)
> dsachs = discretize(sachs, method = "hartemink",
+             breaks = 3, ibreaks = 60, idisc = "quantile")
```

Data are first marginalised in 60 intervals, which are subsequently collapsed while reducing the mutual information between the variables as little as possible. The process stops when each variable has 3 levels (i.e. low, average and high expression).

Discrete Bayesian Networks

Each variable in the `dsachs` data frame is now a factor with three levels (low, average and high concentration). The local distribution of each node is a set of conditional distributions, one for each configuration of the levels of the parents.

```
, , PKC = (1,9.73]

              PKA
praf           (1.95,547] (547,777] (777,4.49e+03]
  (1.61,39.5]  0.3383085 0.2040816      0.2352941
  (39.5,62.6]  0.2885572 0.4897959      0.3921569
  (62.6,552]   0.3731343 0.3061224      0.3725490

, , PKC = (9.73,20.2]

              PKA
praf           (1.95,547] (547,777] (777,4.49e+03]
  (1.61,39.5]  0.3302326 0.3095238      0.3088235
  (39.5,62.6]  0.3023256 0.2857143      0.3823529
  (62.6,552]   0.3674419 0.4047619      0.3088235
```

Posterior Maximisation using **deal**

deal implements learning using a Bayesian approach that supports discrete and mixed data assuming a conditional Gaussian distribution [2]. Structure learning is done with a hill-climbing search maximising the posterior density of the network (as in `hc(..., score = "bde")` in **bnlearn**).

```
> library(deal)
> deal.net = network(dsachs)
> prior = jointprior(deal.net, N = 5)
> deal.net = learn(deal.net, dsachs, prior)$nw
> deal.best = autosearch(deal.net, dsachs, prior)
> bnlearn::fcat(deal::modelstring(deal.best$nw))
```

```
[praf|pmek][pmek][plcg|PIP3][PIP2|plcg:PIP3][PIP3]
[p44.42][pakts473|p44.42][PKA|p44.42:pakts473][PKC|P38]
[P38][pjnk|PKC:P38]
```

Simulated Annealing using **catnet**

catnet [1] learns the network structure in two steps. First, it learns the node ordering from the data using simulated annealing [3],

```
> library(catnet)
> netlist1 = cnSearchSA(dsachs)
```

unless provided by the user.

```
> netlist2 = cnSearchOrder(dsachs, maxParentSet = 5,
+                nodeOrder = sample(names(dsachs)))
```

Then it performs an exhaustive search among the networks with the given node ordering and returns the maximum likelihood estimate.

```
> catnet.best = cnFindBIC(netlist1, nrow(dsachs))
> catnet.best
A catNetwork object with  11  nodes,  1  parents,  3  categories,
 Likelihood =  -9.864328 , Complexity =  50 .
```

PC Algorithm using **pcalg**

pcalg [19] implements the PC algorithm [31], and it is specifically designed to learn causal effects from both discrete and continuous data. **pcalg** can also account for the effects of latent variables through a modified PC algorithm known as Fast Causal Inference (FCI) [31, 4].

```
> library(pcalg)
> suffStat = list(dm = dsachs, nlev = sapply(dsachs, nlevels),
+                 adaptDF = FALSE)
> pcalg.net = pc(suffStat, indepTest = disCItest, p = ncol(dsachs),
+                  alpha = 0.05)
> pcalg.net@graph
A graphNEL graph with undirected edges
Number of Nodes = 11
Number of Edges = 0
```

From the code above, we can also see how to implement custom conditional independence tests and pass them to `pc()` and `fci()` via the `indepTest` argument.

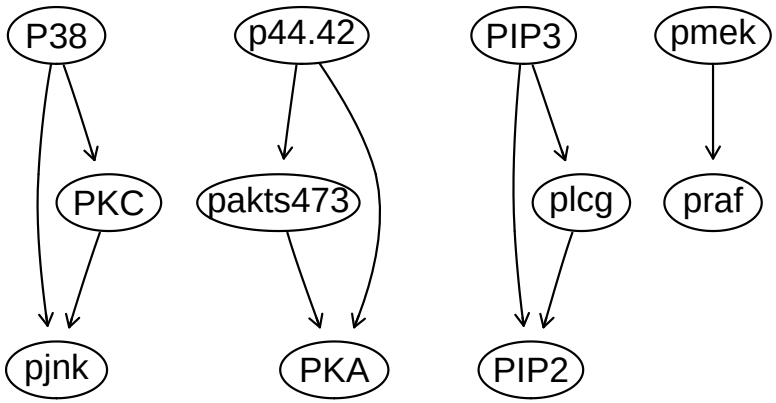
Learning from Ordinal Data with **bnlearn**

The categories in the discretised variables are ordered, so we are discarding information if we assume they come from a multinomial distribution. An appropriate test is the Jonckheere-Terpstra test [8] which will be available soon™ in the next release of **bnlearn**.

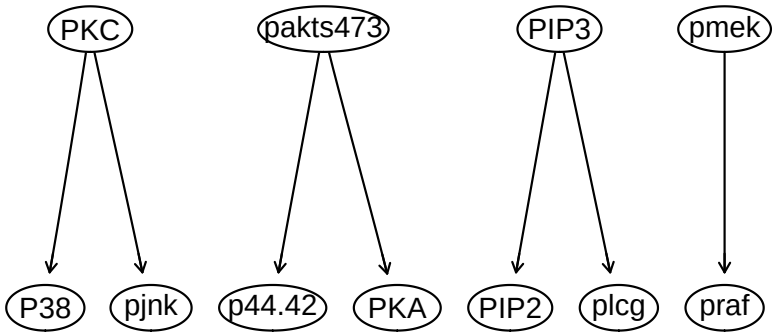
```
> library(bnlearn)
> print(iamb(dsachs, test = "jt"))

  Bayesian network learned via Constraint-based methods
  model:
    [partially directed graph]
  nodes:                                  11
  arcs:                                   8
  [...]

  learning algorithm:                     Incremental Association
  conditional independence test:          Jonckheere-Terpstra Test
  alpha threshold:                        0.05
  tests used in the learning procedure:   223
```

What Kind of Network Structures Did We Learn?

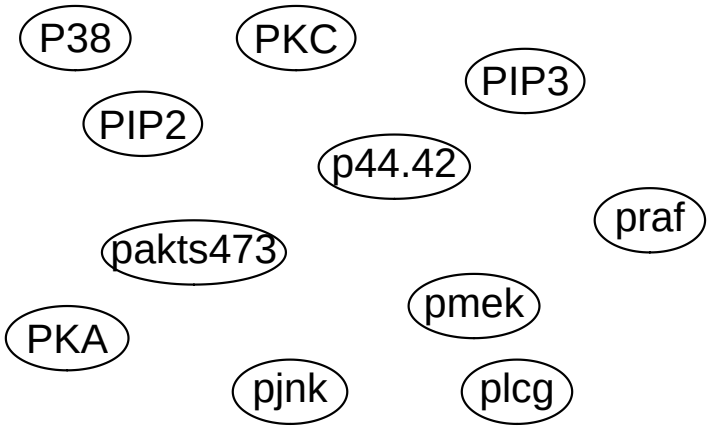
deal package



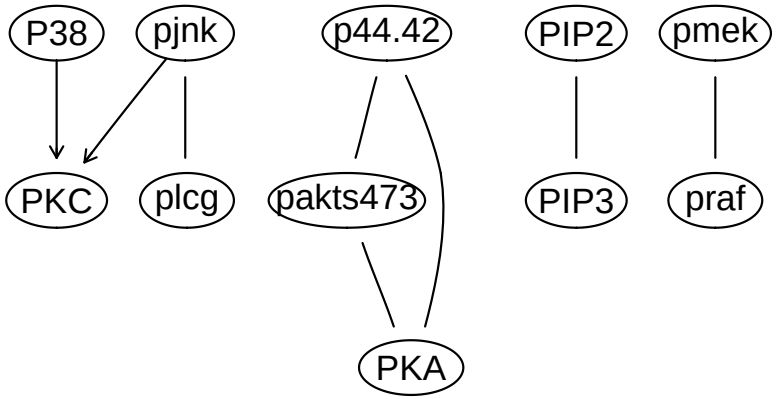
catnet package



pcalg package



bnlearn package



Again, they look nothing like the one validated from literature...

Moving Network Structures Between Packages

- From **bnlearn** to **deal** (and back).

```
> mstring = bnlearn::modelstring(net)
> dnet = deal::network(dsachs[, bnlearn::node.ordering(net)])
> dnet = deal::as.network(bnlearn::modelstring(net), dnet)
> net = bnlearn::model2network(deal::modelstring(dnet))
```

- From **bnlearn** to **pcalg** through **graph** (and back).

```
> pnet = new("pcAlgo", graph = as.graphNEL(net))
> net = bnlearn::as.bn(pnet@graph)
```

- From **bnlearn** to **catnet** (and back).

```
> cnet = cnCatnetFromEdges(nodes = names(dsachs),
+           edges = edges(as.graphNEL(net)))
> net = bnlearn::empty.graph(names(dsachs))
> arcs(net, ignore.cycles = TRUE) = cnMatEdges(cnet)
```

Parameter Learning: Fitting and Modifying

All the packages we covered, with the exception of **bnlearn**, fit the parameters of the network when they learn its structure. As was the case for Gaussian Bayesian networks, in **bnlearn** we can compute the maximum likelihood estimates with

```
> fitted = bn.fit(net, dsachs, method = "mle")
```

and, in addition, the Bayesian posterior estimates with

```
> fitted = bn.fit(net, dsachs, method = "bayes", iss = 5)
```

while controlling the relative weight of the (flat) prior distribution with the `iss` argument. And we can also modify `fitted` or create it from scratch.

```
> new.cpt = matrix(c(0.1, 0.2, 0.3, 0.2, 0.5, 0.6, 0.7, 0.3, 0.1),
+                  dimnames = list(pmek = levels(dsachs$pmek),
+                                  pjnk = levels(dsachs$pjnk)),
+                  byrow = TRUE, ncol = 3)
> fitted$pmek = as.table(new.cpt)
```

Exporting Bayesian Networks to Other Software Packages

bnlearn can export discrete Bayesian networks to software packages such as Hugin, GeNIe or Netica by writing BIF, DSC and NET files.

```
> write.dsc(fitted, file = "bnlearn.dsachs.dsc")
```

Conversely, **bnlearn** can import discrete Bayesian networks created with those software packages by reading the BIF, DSC and NET files they create.

```
> fitted = read.dsc("bnlearn.dsachs.dsc")
```

As an example, that's how a node looks like in a DSC file.

```
node pmek {
  type : discrete [3] = {"[1_21.1]", "[21.1_27.4]", "[27.4_389]"};
}
probability ( pmek | pjnk ) {
  (0) : 0.3622590, 0.2506887, 0.3870523;
  (1) : 0.3828909, 0.1811209, 0.4359882;
  (2) : 0.3763309, 0.2547093, 0.3689599;
}
```

Model Averaging and Interventional Data

Model Averaging

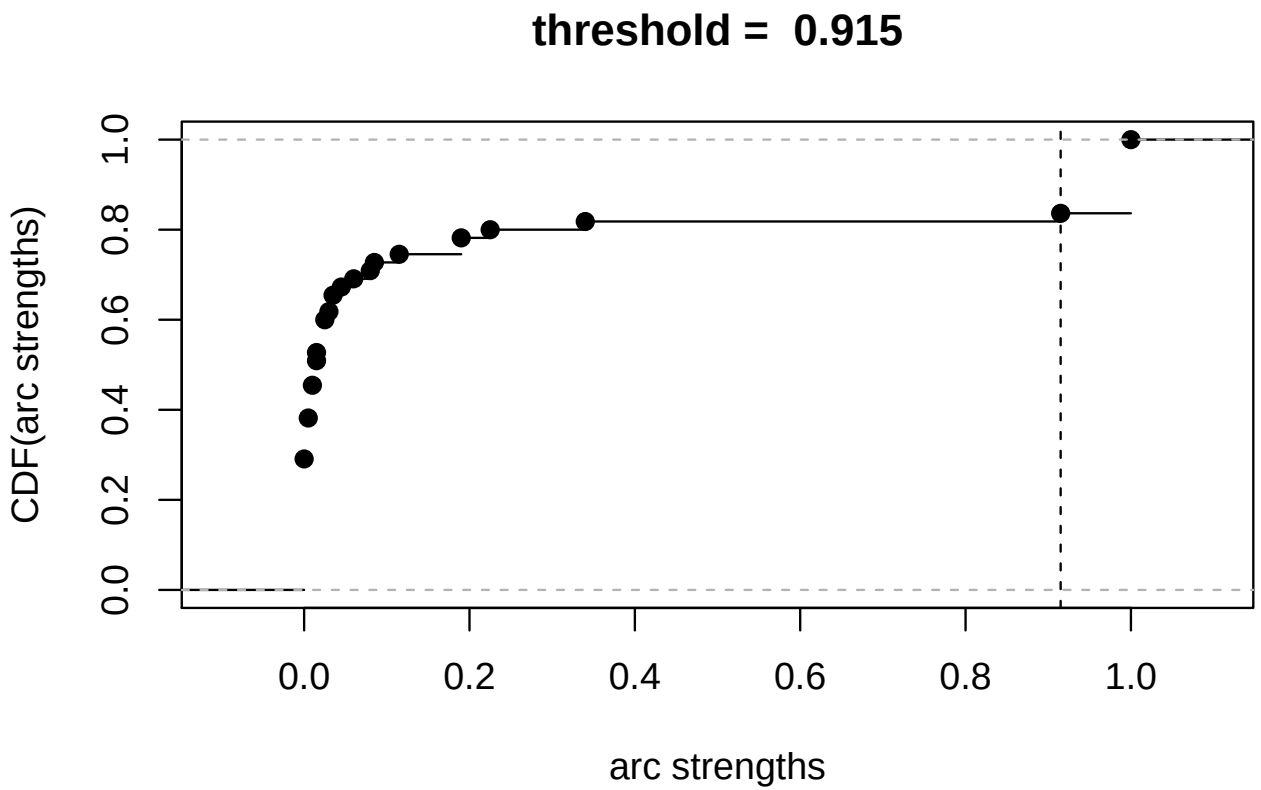
The results of both structure and parameter learning are noisy in most real-world settings, due to limitations in the data and in our knowledge of the processes that control them. Since parameters are learned conditional on the results of structure learning, it's a good idea to use model averaging to obtain a stable network structure from the data. We can generate the networks to average in a few different ways:

- Frequentist: using nonparametric bootstrap and learning one network from each bootstrap sample (aka bootstrap aggregation or bagging) [9].
- Full Bayesian: using Markov Chain Mote Carlo sampling. from the posterior $\mathrm{P}(\mathcal{G} \mid \mathcal{D})$ [10].
- MAP Bayesian: learning a set of network structures with high posterior probability from the original data.

Frequentist Model Averaging: Bootstrap Aggregation

```
> library(bnlearn)
> boot = boot.strength(data = dsachs, R = 200, algorithm = "hc",
+           algorithm.args = list(score = "bde", iss = 10))
> boot[(boot$strength > 0.85) & (boot$direction >= 0.5), ]
        from       to strength direction
1       praf     pmek    1.000 0.5675000
23      plcg     PIP2    0.990 0.5959596
24      plcg     PIP3    1.000 0.9900000
34      PIP2     PIP3    1.000 0.9950000
56    p44.42 pakts473    1.000 0.6175000
57    p44.42      PKA    0.995 1.0000000
67  pakts473      PKA    1.000 1.0000000
89       PKC      P38    1.000 0.5325000
90       PKC     pjnk    1.000 0.9850000
100      P38     pjnk    0.965 1.0000000
> avg.boot = averaged.network(boot, threshold = 0.85)
```

Setting the Threshold



We can use the threshold we learn from the data [30] instead of specifying it with `threshold`, and investigate it with `plot(boot)`.

MAP Bayesian Model Averaging: High-Posterior Networks

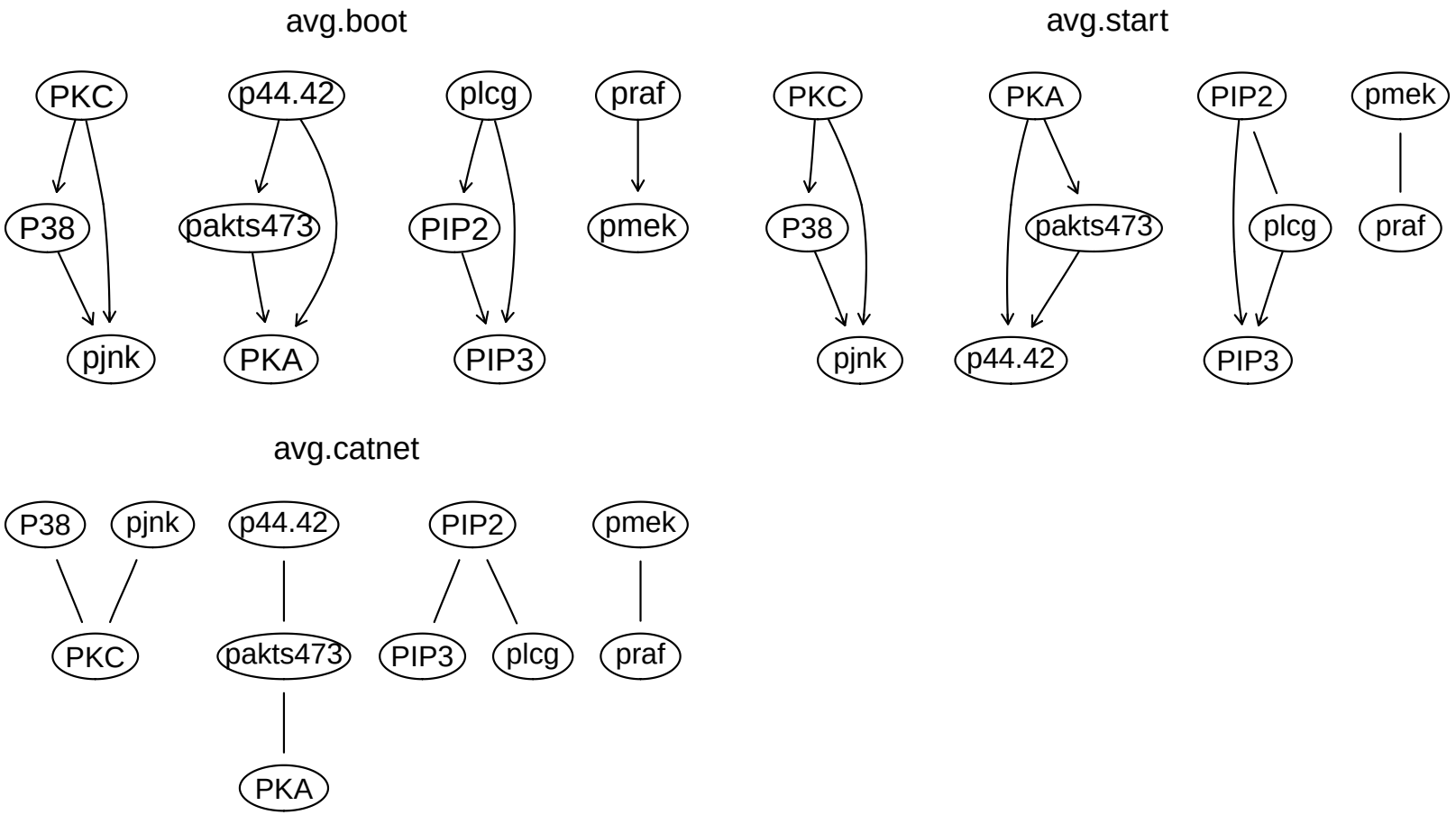
```
> nodes = names(dsachs)
> start = random.graph(nodes = nodes, method = "ic-dag", num = 200)
> netlist = lapply(start, function(net) {
+   hc(dsachs, score = "bde", iss = 10, start = net)
+ })
> rnd = custom.strength(netlist, nodes = nodes)
> head(rnd[(rnd$strength > 0.85) & (rnd$direction >= 0.5), ], n = 9)
       from       to strength direction
1      praf     pmek        1    0.5000
11     pmek     praf        1    0.5000
23     plcg     PIP2        1    0.5000
24     plcg     PIP3        1    1.0000
33     PIP2     plcg        1    0.5000
34     PIP2     PIP3        1    1.0000
66 pakts473   p44.42        1    0.7975
76       PKA   p44.42        1    0.8875
77       PKA pakts473        1    0.5900
> avg.start = averaged.network(rnd, threshold = 0.85)
```

Frequentist Model Averaging: with **catnet**

Structure learning algorithms implemented in other packages can be used for model averaging with `custom.strength()`; the only requirement is that `netlist` must be a list of `bn` objects or a list of arc sets stored in 2-columns matrices (like the ones returned by the `arcs()` function).

```
> library(catnet)
> netlist = vector(200, mode = "list")
> ndata = nrow(dsachs)
> netlist = lapply(netlist, function(net) {
+   boot = dsachs[sample(ndata, replace = TRUE), ]
+   nets = cnSearchOrder(boot)
+   best = cnFindBIC(nets, ndata)
+   cnMatEdges(best)
+ })
> sa = custom.strength(netlist, nodes = nodes)
> avg.catnet = averaged.network(sa, threshold = 0.85)
```

Averaged Bayesian Networks



The structure is stable overall, but the networks are still quite different from the validated network...

Modelling Interventions

In most data sets, all observations are collected under the same general conditions and can be modelled with a single Bayesian network, because they follow the same probability distribution.

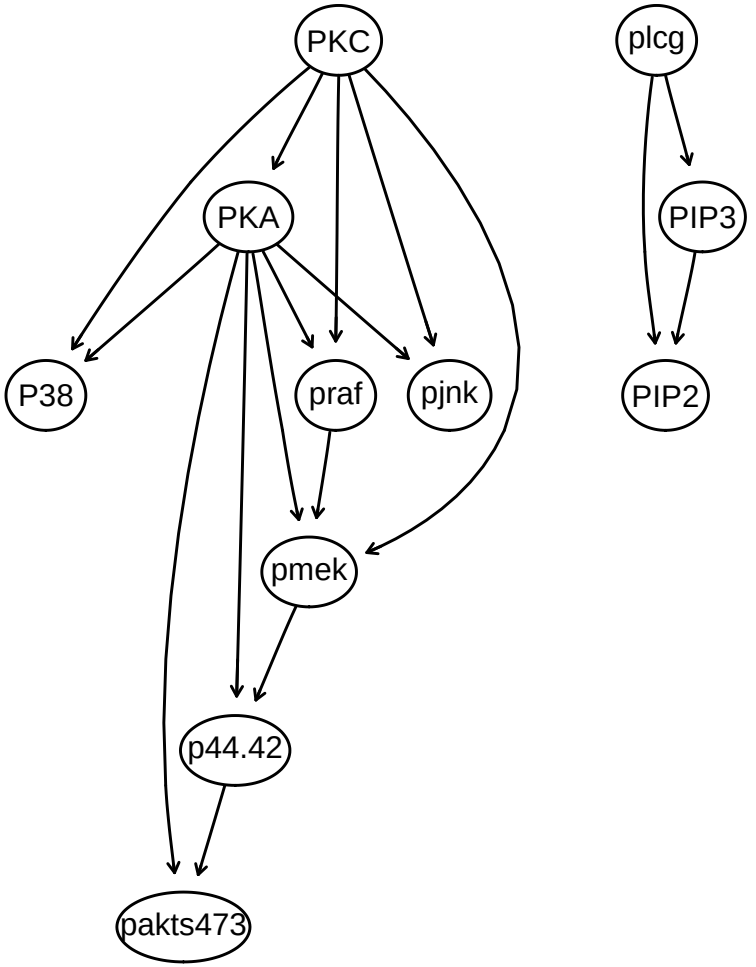
However, this is not the case when samples from different experiments are analysed together with a single, encompassing model. In addition to the data set we have analysed so far, which is subject only to a general stimulus meant to activate the desired paths, we have 9 other data sets subject to different targeted stimolatory cues and inhibitory interventions.

```
> isachs = read.table("sachs.interventional.txt", header = TRUE,
+             colClasses = "factor")
```

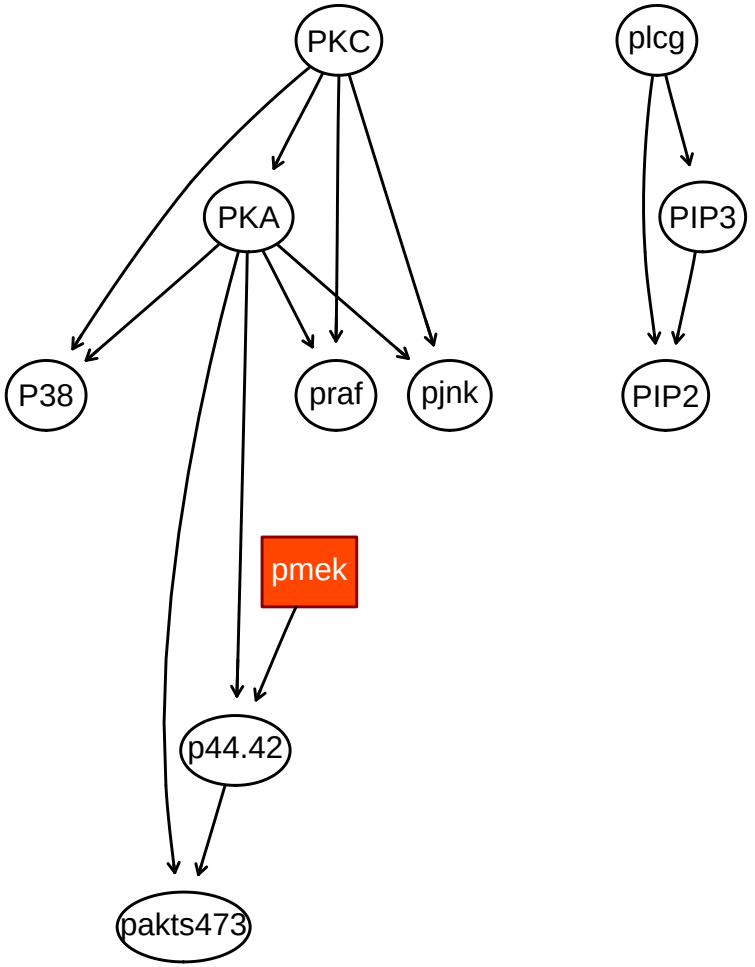
Such data are often called interventional, because the values of specific variables in the model are set by an external intervention of the investigator.

Modelling an Intervention on pmek

model without interventions



model with interventions



Modelling Intervention with an Extra Node

One intuitive way to model these data sets with a single, encompassing Bayesian network is to include the intervention `INT` in the network and to make all variables depend on it with a `whitelist`.

```
> wh = matrix(c(rep("INT", 11), names(isachs)[1:11]), ncol = 2)
> bn.wh = tabu(isachs, whitelist = wh, score = "bde",
+            iss = 10, tabu = 50)
```

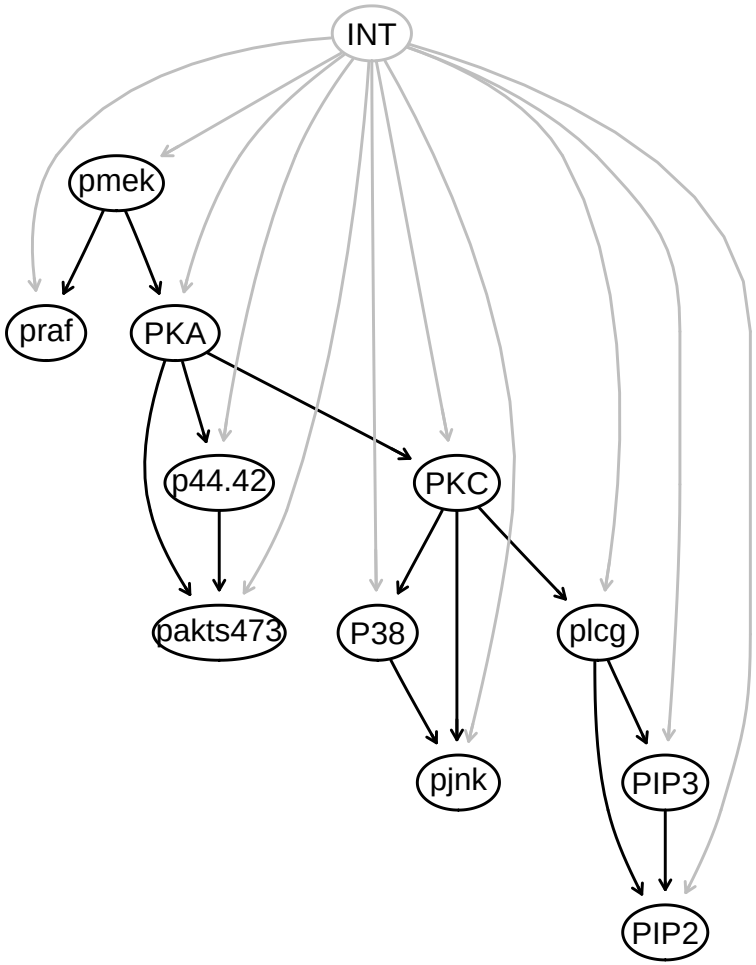
We can also let the structure learning algorithm decide which arcs connecting `INT` to the other nodes should be included in the network, and blacklist all the arcs towards `INT`.

```
> tiers = list("INT", names(isachs)[1:11])
> bl = tiers2blacklist(nodes = tiers)
> bn.tiers = tabu(isachs, blacklist = bl,
+                 score = "bde", iss = 10, tabu = 50)
```

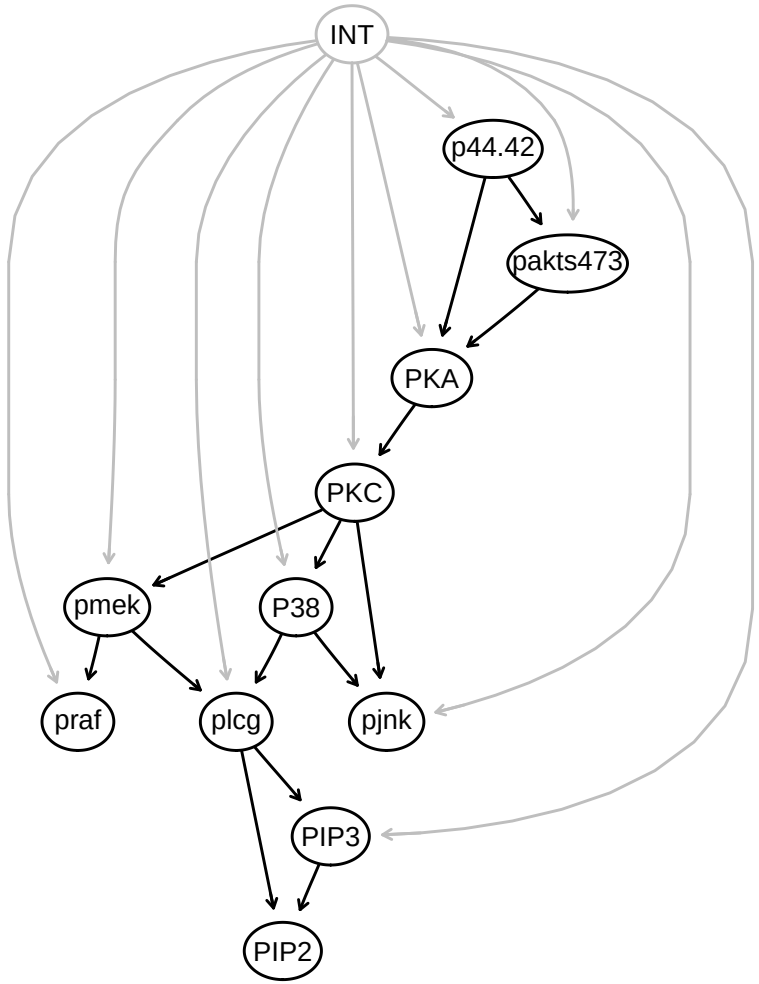
`tiers2blacklist()` builds a `blacklist` such that all arcs going from a node in a particular element of the `nodes` argument to a node in one of the previous elements are blacklisted.

Modelling Intervention with an Extra Node

All nodes depend on INT



Relevant nodes depend on INT



Adapting Posterior Probability Estimates

A better solution is to remove INT from the data,

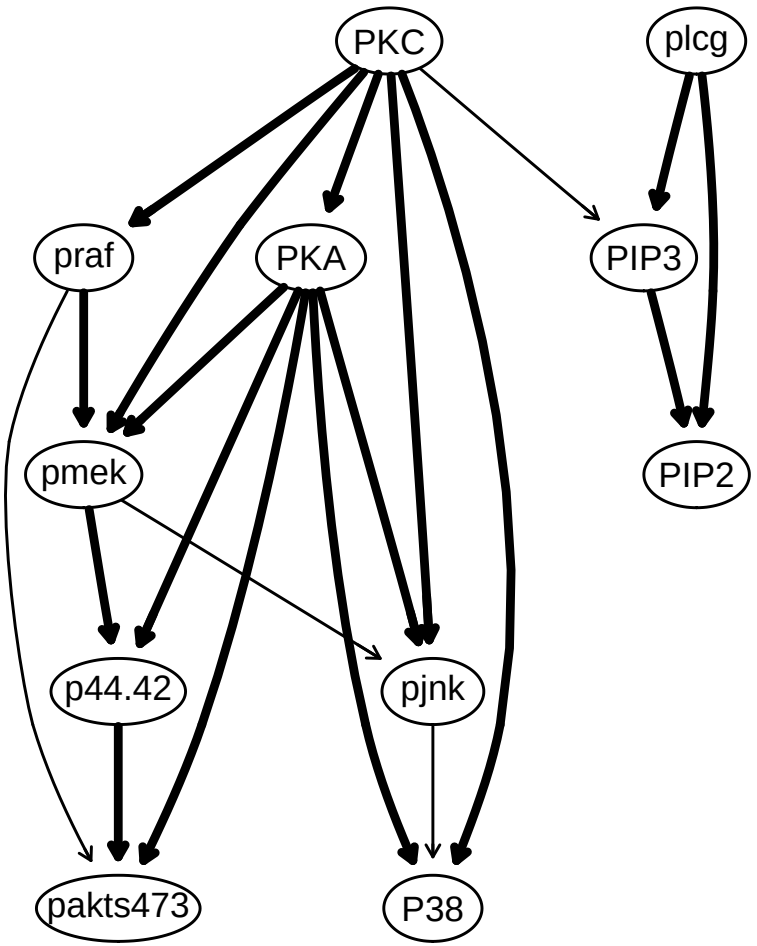
```
> INT = sapply(1:11, function(x) {
+                                 which(isachs$INT == x) })
> isachs = isachs[, 1:11]
> nodes = names(isachs)
> names(INT) = nodes
```

and incorporate it into structure learning using a modified posterior probability score (mbde) that takes its effect into account [5].

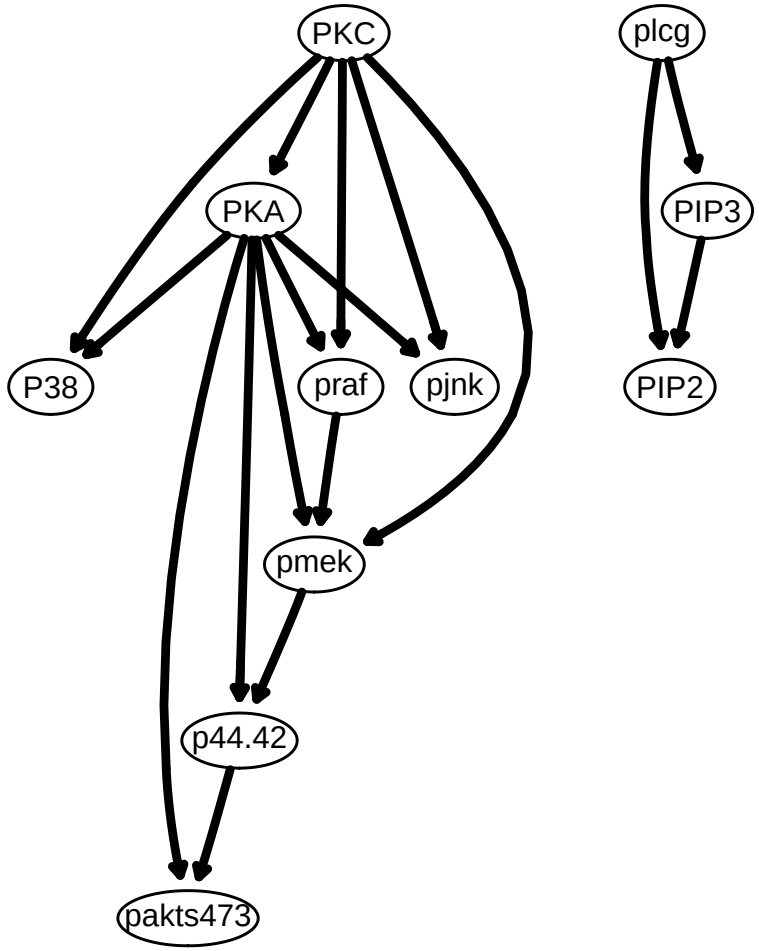
```
> start = random.graph(nodes = nodes,
+    method = "melancon", num = 200, burn.in = 10^5,
+    every = 100)
>  netlist = lapply(start, function(net) {
+    tabu(isachs, score = "mbde", exp = INT,
+      iss = 1, start = net, tabu = 50) })
>  arcs = custom.strength(netlist, nodes = nodes, cpdag = FALSE)
>  bn.mbde = averaged.network(arcs, threshold = 0.85)
```

Modelling Intervention with an Extra Node

bn.mbde



validated network



Comparing Network Structures

When we compare two Bayesian networks, it is important to compare their equivalence classes through the respective CPDAGs instead of the networks themselves.

```
> learned.spec = paste("[plcg][PKC][praf|PKC][PIP3|plcg:PKC]",
+   "[PKA|PKC][pmek|praf:PKA:PKC][PIP2|plcg:PIP3][p44.42|PKA:pmek]",
+   "[pakts473|praf:p44.42:PKA][pjnk|pmek:PKA:PKC][P38|PKA:PKC:pjnk]")
> true.spec = paste("[PKC][PKA|PKC][praf|PKC:PKA][pmek|PKC:PKA:praf]",
+   "[p44.42|pmek:PKA][pakts473|p44.42:PKA][P38|PKC:PKA]",
+   "[pjnk|PKC:PKA][plcg][PIP3|plcg][PIP2|plcg:PIP3]")
> true = model2network(true.spec)
> learned = model2network(learned.spec)
> unlist(compare(true, learned))

tp fp fn
16  4  1

> unlist(compare(cpdag(true), cpdag(learned)))

tp fp fn
14  6  3
```

Inference

Inference in the Sachs et al. Paper

In their paper, Sachs et al. used the validated network to substantiate two claims:

1. a direct perturbation of `p44.42` should influence `pakts473`;
2. a direct perturbation of `p44.42` should not influence `PKA`.

The probability distributions of `p44.42`, `pakts473` and `PKA` were then compared with the results of two ad-hoc experiments to confirm the validity and the direction of the inferred causal influences.

```
> for (i in names(isachs))
+   levels(isachs[, i]) = c("LOW", "AVG", "HIGH")
> fitted = bn.fit(true, isachs, method = "bayes")
```

For convenience, we rename the levels of each variable to `LOW`, `AVERAGE` and `HIGH`.

Exact Inference with **gRain**

gRain [18] implements exact inference for discrete Bayesian networks via junction tree belief propagation [21]. We can export a network fitted with **bnlearn**,

```
> library(gRain)
> jtree = compile(as.grain(fitted))
```

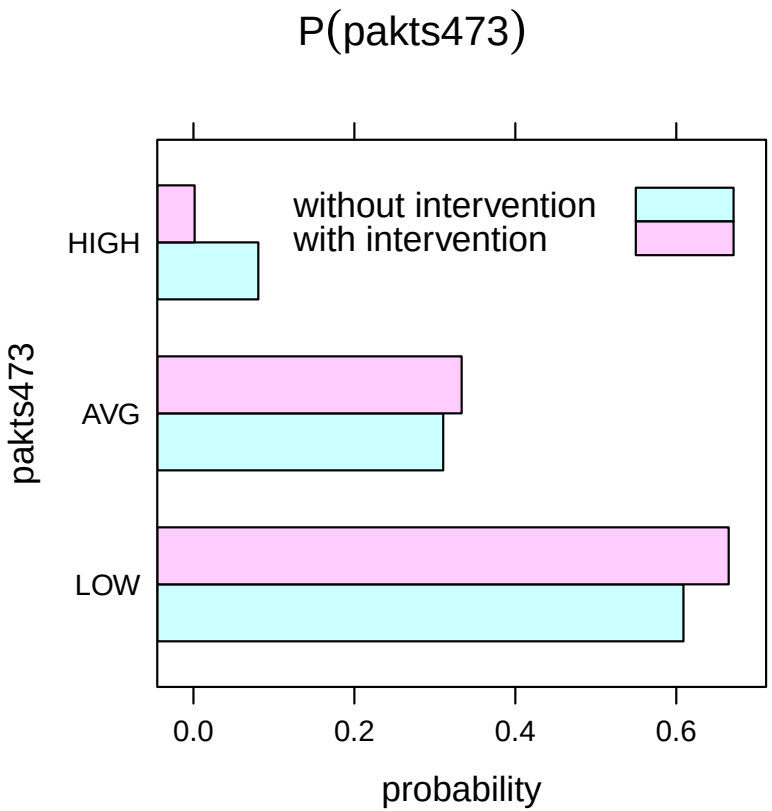
set the evidence (i.e. the event we condition on),

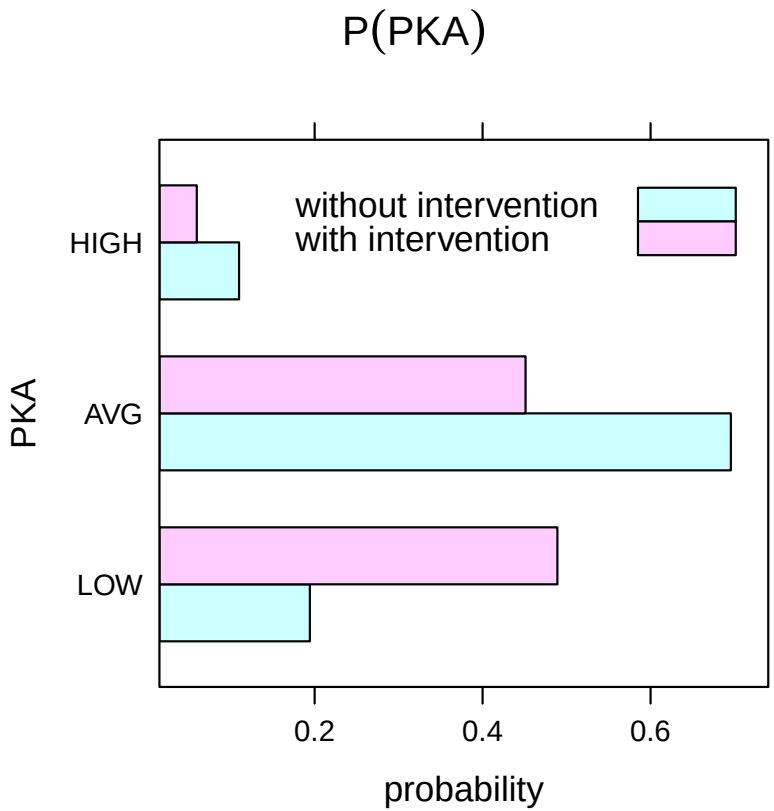
```
> jprop = setFinding(jtree, nodes = "p44.42", states = "LOW")
```

and compare conditional and unconditional probabilities.

```
> querygrain(jtree, nodes = "pakts473")$pakts473
pakts473
       LOW        AVG       HIGH
0.60893407 0.31041282 0.08065311
> querygrain(jprop, nodes = "pakts473")$pakts473
pakts473
        LOW         AVG        HIGH
0.665161776 0.333333333 0.001504891
```

Graphical Comparison of Probability Distributions





Causal and non-causal use of Bayesian networks are different...

Approximate Inference with **bnlearn**

bnlearn implements approximate inference via rejection sampling (called logic sampling in this setting), and soon™ via importance sampling (likelihood weighting) [22]. `cpdist` generates random observations from `fitted` for the nodes `nodes` conditional on the evidence `evidence`.

```
> particles = cpdist(fitted, nodes = "pakts473",
+                    evidence = (p44.42 == "LOW"))
> prop.table(table(particles))
particles
        LOW         AVG        HIGH
0.665686790 0.332884451 0.001428759
```

On the other hand, `cpquery` returns the probability of a specific `event`.

```
> cpquery(fitted, event = (pakts473 == "AVG"),
+   evidence = (p44.42 == "LOW"))
[1] 0.3319946
```

Both can be configured to generate samples in parallel (using the **snow** or **parallel** packages) and/or in small batches to fit available memory.

Approximate Inference with **bnlearn**

Compared to exact inference in **gRain**, approximate inference in **bnlearn** often requires more memory and is much slower. However, it is more flexible and allows much more complicated queries.

```
> cpquery(fitted,
+   event = (pakts473 == "LOW") & (PKA != "HIGH"),
+   evidence = (p44.42 == "LOW") | (praf == "LOW"))
[1] 0.5593692
> cpdist(fitted, n = 6, nodes = nodes(fitted),
+   evidence = (p44.42 == "LOW") | (praf == "LOW") &
+              (pakts473 %in% c("LOW", "HIGH")))
   P38 p44.42 pakts473 PIP2 PIP3 pjnk  PKA PKC plcg pmek praf
1  AVG    AVG      LOW  LOW  AVG  LOW HIGH LOW  LOW  LOW  LOW
2  LOW    AVG      LOW  LOW  AVG  AVG  AVG AVG  LOW  LOW  LOW
3 HIGH    LOW      LOW  LOW  LOW  AVG  LOW LOW  LOW  LOW HIGH
```

We can also generate random observations from the unconditional distribution with `cpdist(fitted, n = 6, TRUE, TRUE)` or `rbn(fitted, n = 6)` as a term of comparison.

Thanks for attending!

References

References I

[1] N. Balov and P. Salzman.
catnet: Categorical Bayesian Network Inference, 2012.
R package version 1.13.4.

[2] S. G. Bøttcher and C. Dethlefsen.
deal: A Package for Learning Bayesian Networks.
Journal of Statistical Software, 8(20):1–40, 2003.

[3] V. Černý.
Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm.
Journal of Optimization Theory and Applications, 45(1):41–51, 1985.

[4] D. Colombo, M. H. Maathuis, M. Kalish, and T. S. Richardson.
Learning High-Dimensional Directed Acyclic Graphs with Latent and Selection Variables.
Annals of Statistics, 40(1):294–321, 2012.

[5] G. F. Cooper and C. Yoo.
Causal Discovery from a Mixture of Experimental and Observational Data.
In *UAI '99: Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence*, pages 116–125. Morgan Kaufmann, 1995.

References II

[6] G. Csardi and T. Nepusz.
The igraph Software Package for Complex Network Research.
InterJournal, Complex Systems, 1695:1–38, 2006.

[7] D. Dor and M. Tarsi.
A Simple Algorithm to Construct a Consistent Extension of a Partially Oriented Graph.
Technical report, UCLA, Cognitive Systems Laboratory, 1992.
Available as Technical Report R-185.

[8] D. I. Edwards.
Introduction to Graphical Modelling.
Springer, 2nd edition, 2000.

[9] N. Friedman, M. Goldszmidt, and A. Wyner.
Data Analysis with Bayesian Networks: A Bootstrap Approach.
In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 206–215. Morgan Kaufmann, 1999.

References III

[10] N. Friedman and D. Koller.
Being Bayesian about Bayesian Network Structure: A Bayesian Approach to Structure Discovery in Bayesian Networks.
Machine Learning, 50(1–2):95–126, 2003.

[11] N. Friedman, D. Pe'er, and I. Nachman.
Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm.
In *Proceedings of 15th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 206–215. Morgan Kaufmann, 1999.

[12] D. Geiger and D. Heckerman.
Learning Gaussian Networks.
Technical report, Microsoft Research, Redmond, Washington, 1994.
Available as Technical Report MSR-TR-94-10.

[13] R. Gentleman, E. Whalen, W. Huber, and S. Falcon.
graph: A package to handle graph data structures.
R package version 1.37.7.

References IV

[14] J. Gentry, L. Long, R. Gentleman, S. Falcon, F. Hahne, D. Sarkar, and K. D. Hansen.
Rgraphviz: Provides plotting capabilities for R graph objects.
R package version 2.3.8.

[15] J. J. Goeman.
penalized R package, 2012.
R package version 0.9-41.

[16] A. J. Hartemink.
Principled Computational Methods for the Validation and Discovery of Genetic Regulatory Networks.
PhD thesis, School of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2001.

[17] D. Heckerman, D. Geiger, and D. M. Chickering.
Learning Bayesian Networks: The Combination of Knowledge and Statistical Data.
Machine Learning, 20(3):197–243, September 1995.
Available as Technical Report MSR-TR-94-09.

References V

[18] Søren Højsgaard.
Graphical Independence Networks with the gRain Package for R.
Journal of Statistical Software, 46(10):1–26, 2012.

[19] M. Kalisch, M. Mächler, D. Colombo, M. H. Maathuis, and P. Bühlmann.
Causal Inference Using Graphical Models with the R Package pcalg.
Journal of Statistical Software, 47(11):1–26, 2012.

[20] R. Kohavi and M. Sahami.
Error-Based and Entropy-Based Discretization of Continuous Features.
In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD '96)*, pages 114–119. AAAI Press, 1996.

[21] D. Koller and N. Friedman.
Probabilistic Graphical Models: Principles and Techniques.
MIT Press, 2009.

[22] K. Korb and A. Nicholson.
Bayesian Artificial Intelligence.
Chapman and Hall, 2nd edition, 2010.

References VI

[23] P. Larrañaga, B. Sierra, M. J. Gallego, M. J. Michelena, and J. M. Picaza.
Learning Bayesian Networks by Genetic Algorithms: A Case Study in the Prediction of Survival in Malignant Skin Melanoma.
In *Proceedings of the 6th Conference on Artificial Intelligence in Medicine in Europe (AIME '97)*, pages 261–272. Springer, 1997.

[24] D. Margaritis.
Learning Bayesian Network Model Structure from Data.
PhD thesis, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 2003.
Available as Technical Report CMU-CS-03-153.

[25] R. Nagarajan, M. Scutari, and S. Lèbre.
Bayesian Networks in R with Applications in Systems Biology.
Use R! series. Springer, 2013.

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

References VII

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

[28] S. J. Russell and P. Norvig.
Artificial Intelligence: A Modern Approach.
Prentice Hall, 3rd edition, 2009.

[29] K. Sachs, O. Perez, D. Pe'er, D. A. Lauffenburger, and G. P. Nolan.
Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data.
Science, 308(5721):523–529, 2005.

[30] M. Scutari and R. Nagarajan.
On Identifying Significant Edges in Graphical Models of Molecular Networks.
Artificial Intelligence in Medicine, 57(3):207–217, 2013.
Special Issue containing the Proceedings of the Workshop "Probabilistic Problem Solving in Biomedicine" of the 13th Artificial Intelligence in Medicine (AIME) Conference, Bled (Slovenia), July 2, 2011.

References VIII

[31] P. Spirtes, C. Glymour, and R. Scheines.
Causation, Prediction, and Search.
MIT Press, 2nd edition, 2001.

[32] I. Tsamardinos, C. F. Aliferis, and A. Statnikov.
Algorithms for Large Scale Markov Blanket Discovery.
In *Proceedings of the 16th International Florida Artificial Intelligence Research Society Conference*, pages 376–381. AAAI Press, 2003.

[33] I. Tsamardinos, L. E. Brown, and C. F. Aliferis.
The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm.
Machine Learning, 65(1):31–78, 2006.

[34] W. N. Venables and B. D. Ripley.
Modern Applied Statistics with S.
Springer, 4th edition, 2002.

[35] K. Y. Yeung, C. Fraley, A. Murua, A. E. Raftery, and W. L. Ruzzo.
Model-Based Clustering and Data Transformations for Gene Expression Data.
Bioinformatics, 17(10):977–987, 2001.