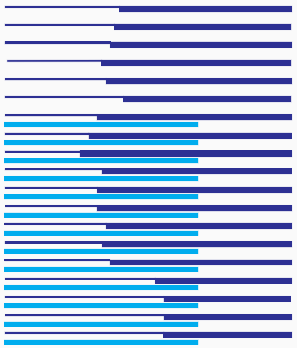


Advanced Probabilistic Modelling

Models and Distributional Assumptions

Marco Scutari

Dalle Molle Institute for
Artificial Intelligence (IDSIA)

Academic Year 2025–2026

What About the Probability Distributions?

The second component of a BN is the probability distribution $\mathrm{P}(\mathbf{X})$. The choice should be such that the BN:

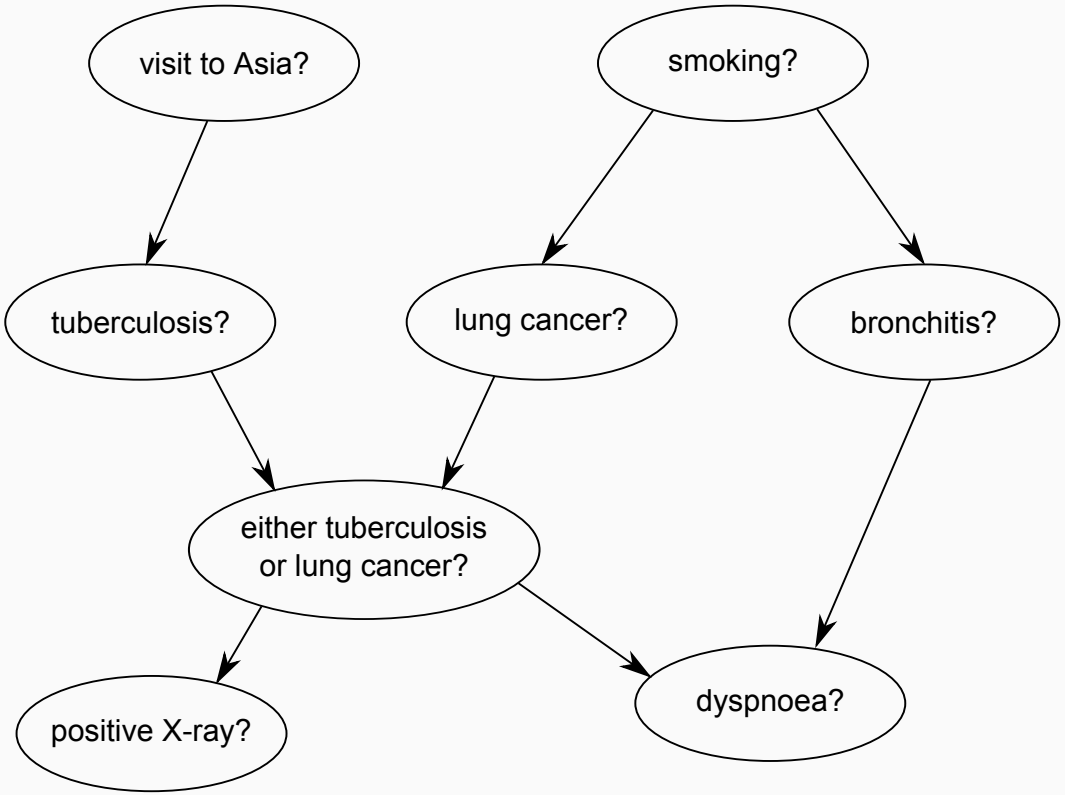
- can be learned efficiently from data;
- is flexible (distributional assumptions should not be too limiting);
- is easy to query to perform inference.

The three most common choices in the literature (by far), are:

- discrete BNs (DBNs), in which $\mathbf{X}$ and the $X_i \mid \Pi_{X_i}$ are multinomial;
- Gaussian BNs (GBNs), in which $\mathbf{X}$ is multivariate normal and the $X_i \mid \Pi_{X_i}$ are univariate normal;
- conditional linear Gaussian BNs (CGBNs), in which X is a mixture of multivariate normals and the $X_i \mid \Pi_{X_i}$ are either multinomial, univariate normal or mixtures of normals.

It has been proved in the literature that exact inference is possible in these three cases, hence their popularity.

Discrete BNs



A classic example of DBN is the ASIA network from Lauritzen & Spiegelhalter (1988), which includes a collection of binary variables. It describes a simple diagnostic problem for tuberculosis and lung cancer.

Total parameters of $\mathbf{X}$:
$2^8 - 1 = 255$

The Global Distribution

The global distribution $\mathrm{P}(\mathbf{X})$ is

$$\mathbf{X} \sim Mul\left(\pi_{\text{<insert one subscript for each node here>}}\right).$$

The parameters $\pi_{\text{<insert one subscript for each node here>}}$ describe the probability of each possible combination of the values of the variables in $\mathbf{X}$.

Problems:

- There are too many parameters: if each variable can take l values $|\Theta| = |\mathbf{X}|^l$ - 1! It is difficult to even tabulate them, much less give them a practical interpretation.
- They are too small: $\approx 1/(|\mathbf{X}|^l)$ on average since they sum up to 1.
- They have no structure: they do not separate main effects, low-order interactions and high-order interactions.

The Local Distributions

The definition of a BN states that $\mathrm{P}(\mathbf{X}) = \prod_{i=1}^{N} \mathrm{P}(X_i \,|\, \Pi_{X_i})$.
Let's assume:

1. Positivity: all probabilities are strictly positive.
2. Parameter independence: probabilities in each $X_i \,|\, \Pi_{X_i}$ that are conditional on different parent configurations are independent.
3. Parameter modularity: probabilities in different $X_i \,|\, \Pi_{X_i}$ are independent.

Then $X_i \,|\, \Pi_{X_i} \sim Mul\,(\pi_{ik\,|\,j})$ where $\pi_{ik\,|\,j} = \mathrm{P}(X_i = k \,|\, \Pi_{X_i} = j)$.

Problems we solved:

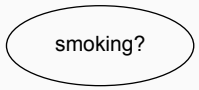
- Each local distribution has only $|\Theta_{X_i}| = (|\Pi_{X_i}| + 1)^l$ parameters.
- Each $\pi_{ik\,|\,j}$ is $\approx 1/l$ on average.

Problems we did not solve:

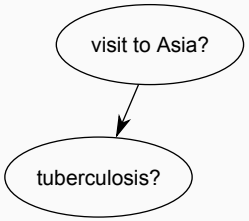
- The $\pi_{ik\,|\,j}$ still do not separate the effects of the Π_{X_i}.

Conditional Probability Tables (CPTs)

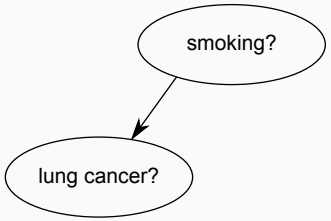
yes	no
0.01	0.99



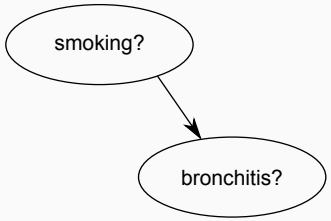
yes	no
0.50	0.50



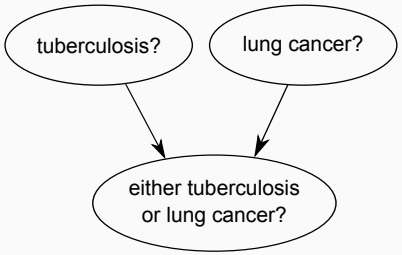
	yes	no
yes	0.05	0.95
no	0.01	0.99



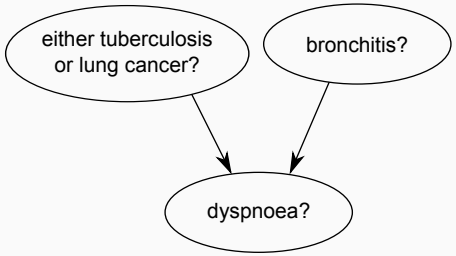
	yes	no
yes	0.10	0.90
no	0.01	0.99



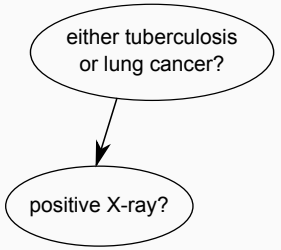
	yes	no
yes	0.60	0.40
no	0.30	0.70



	yes	no
yes:yes	1	0
yes:no	1	0
no:yes	1	0
no:no	0	1



	yes	no
yes:yes	0.90	0.10
yes:no	0.70	0.30
no:yes	0.80	0.20
no:no	0.10	0.90



	yes	no
yes	0.98	0.02
no	0.05	0.95

The local distributions $X_i \mid \Pi_{X_i}$ take the form of conditional probability tables for each node given all the configurations of the values of its parents.

Overall parameters of the $X_i \mid \Pi_{X_i}$: 18

BNLEARN: Creating a Discrete BN (ASIA)

```
asia.dag = model2network("[A][S][T|A][L|S][B|S][D|B:E][E|T:L][X|E]")

lv = c("yes", "no")

A.prob = array(c(0.01, 0.99), dim = 2, dimnames = list(A = lv))
S.prob = array(c(0.01, 0.99), dim = 2, dimnames = list(A = lv))
T.prob = array(c(0.05, 0.95, 0.01, 0.99), dim = c(2, 2),
           dimnames = list(T = lv, A = lv))
L.prob = array(c(0.1, 0.9, 0.01, 0.99), dim = c(2, 2),
          dimnames = list(L = lv, S = lv))
B.prob = array(c(0.6, 0.4, 0.3, 0.7), dim = c(2, 2),
          dimnames = list(B = lv, S = lv))
D.prob = array(c(0.9, 0.1, 0.7, 0.3, 0.8, 0.2, 0.1, 0.9), dim = c(2, 2, 2),
          dimnames = list(D = lv, B = lv, E = lv))
E.prob = array(c(1, 0, 1, 0, 1, 0, 0, 1), dim = c(2, 2, 2),
          dimnames = list(E = lv, T = lv, L = lv))
X.prob = array(c(0.98, 0.02, 0.05, 0.95), dim = c(2, 2),
          dimnames = list(X = lv, E = lv))

cpt = list(A = A.prob, S = S.prob, T = T.prob, L = L.prob, B = B.prob,
        D = D.prob, E = E.prob, X = X.prob)
asia.bn = custom.fit(asia.dag, cpt)
```

bnlearn: Conditional Probability Tables (I)

Smoking:

```
asia.bn$S
```

```
  Parameters of node S (multinomial distribution)

Conditional probability table:
 A
 yes   no
0.01 0.99
```

Lung cancer:

```
asia.bn$L
```

```
  Parameters of node L (multinomial distribution)

Conditional probability table:

     S
L      yes   no
  yes 0.10 0.01
  no  0.90 0.99
```

BNLEARN: CONDITIONAL PROBABILITY TABLES (II)

Dyspnoea:

```
asia.bn$D
```

```
  Parameters of node D (multinomial distribution)

Conditional probability table:

, , E = yes

     B
D     yes  no
  yes 0.9 0.7
  no  0.1 0.3

, , E = no

     B
D     yes  no
  yes 0.8 0.1
  no  0.2 0.9
```

Conditional Probability Tables and Logistic Regression

A conditional probability table $\mathrm{P}(X_i \mid \Pi_{X_i})$ is equivalent to a saturated multinomial logistic regression for X_i with all possible interaction terms (of all possible orders) for the Π_{X_i}: a model that has as many covariates as there are free probabilities in the conditional probability table.

For each value k of X_i, via the multinomial link function:

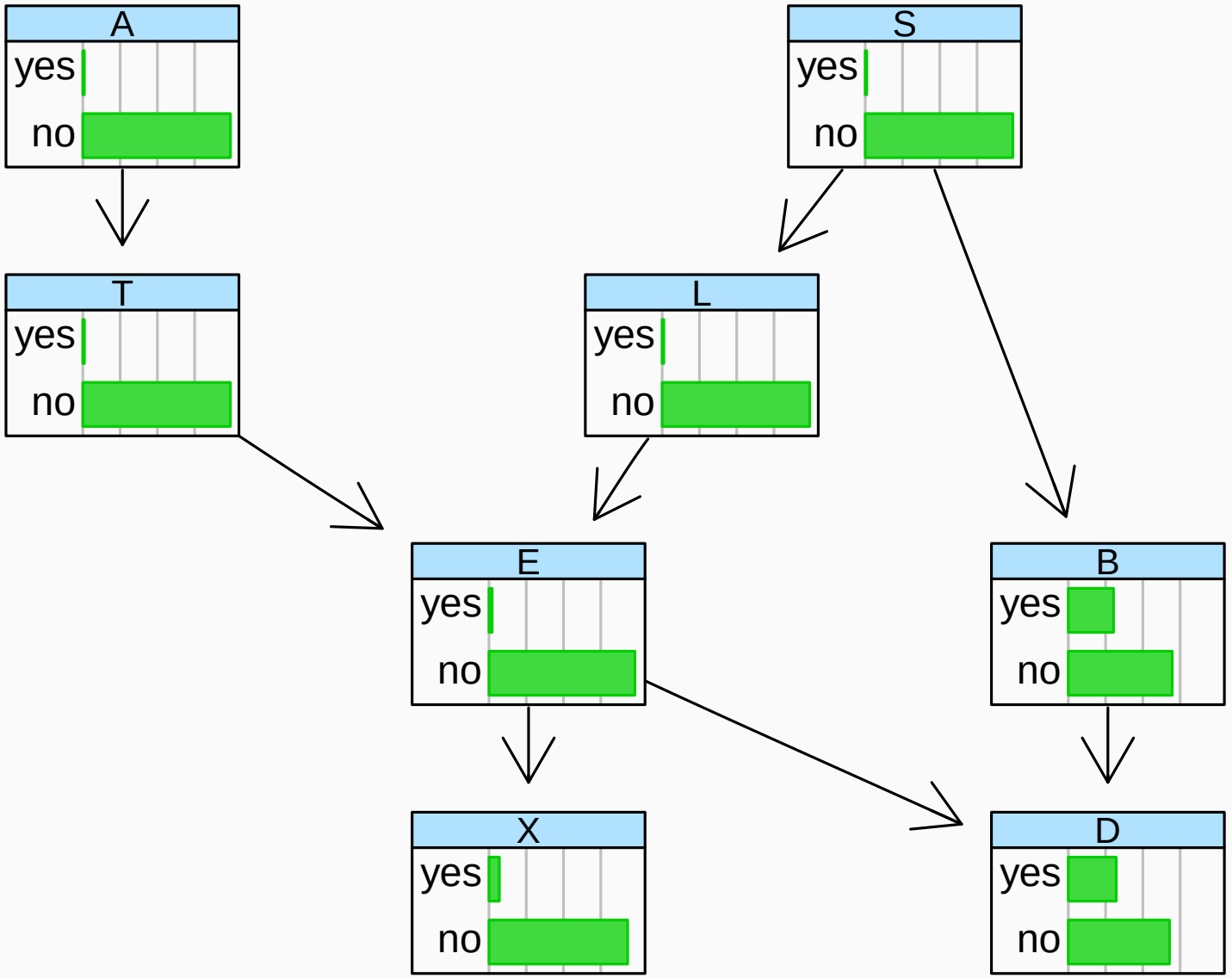
$$\log(\pi_{ik\,|\,\bullet}/\pi_{i1\,|\,\bullet}) = \eta_{ik\,|\,\bullet} \qquad\qquad \eta_{ik\,|\,\bullet} = \mathbf{Z}_i\boldsymbol{\gamma}_{ik}$$

where $\mathbf{Z}_i$ is the design matrix for the saturated model and the $\boldsymbol{\gamma}_{ik}$ are the regression coefficients.

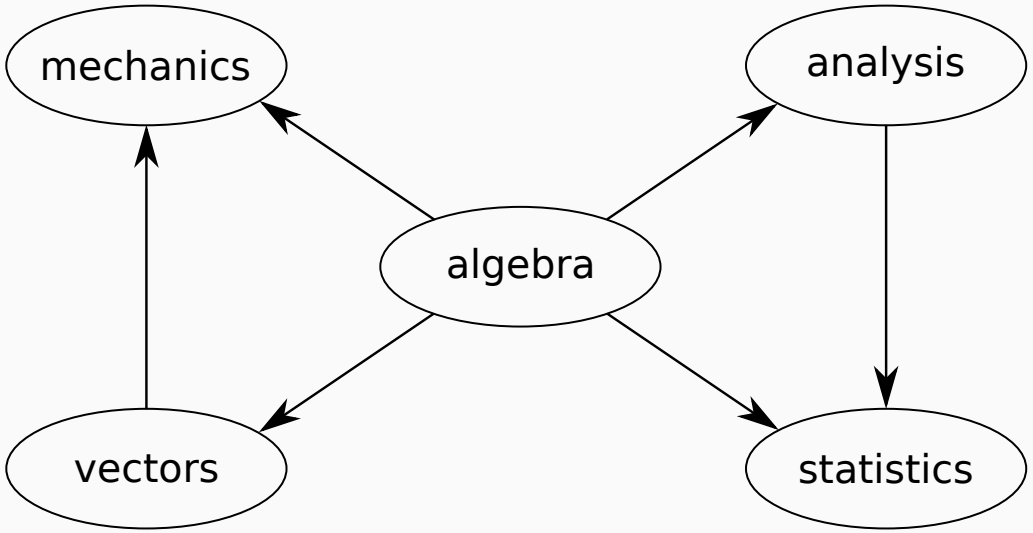
Now we can:

- separate main and interaction effects; and
- simplify the structure of Θ_{X_i} by removing higher-order interactions as needed.

Canonical Plot of a DBN



Gaussian BNs



A classic example of GBN is the MARKS network from Mardia, Kent & Bibby (1979), which describes the relationships between the marks on 5 math-related topics.

Total parameters of $\mathbf{X}$: $5 + 15 = 20$

The global distribution $\mathrm{P}(\mathbf{X})$ is

$$\mathbf{X} \sim N(\boldsymbol{\mu}, \Sigma)$$

where $\boldsymbol{\mu}$ is the vector of the means ($|\mathbf{X}| \times 1$) and Σ is the covariance matrix ($|\mathbf{X}| \times |\mathbf{X}|$).

Classical multivariate statistics tells us that

1. if we assume that Σ is full rank, we can compute the precision matrix $\Omega = \Sigma^{-1}$;
2. if $\Omega_{ij} = 0$ then $X_i \perp\!\!\!\perp_P X_j \,|\, \mathbf{X}\ \{X_i, X_j\}$.

In a GBN, the absence of an arc $X_i \to X_j$ in the DAG implies $X_i \perp\!\!\!\perp_G X_j \,|\, \mathbf{X}\ \{X_,X_j\}$, which in turn implies $X_i \perp\!\!\!\perp_P X_j \,|\, \mathbf{X}\ \{X_,X_j\}$.

As a result, GBNs can only model linear dependencies.

Partial Correlations and Local Distributions

The local distributions $\mathrm{P}(X_i \mid \Pi_{X_i})$ have two equivalent parametrisations:

- using partial correlations;
- using linear regressions.

The partial correlation between X_i and each of the Π_{X_i} given the others can be computed from Σ:

1. subset Σ to $\Sigma^{(i)} = \Sigma_{\{X_i, \Pi_{X_i}\}}$;
2. compute $\Omega^{(i)}$, the inverse of $\Sigma^{(i)}$;
3. the partial correlation is $\rho_{ij} = -\Omega^{(i)}_{1,j} / \sqrt{\Omega^{(i)}_{1,1} / \Omega^{(i)}_{j,j}}$.

The equivalent coefficients in the linear regression model

$$X_i = \mu_{X_i} + \Pi_{X_i} \boldsymbol{\beta}_{X_i} + \varepsilon_{X_i}, \qquad \varepsilon_{X_i} \sim N(0, \sigma^2_{X_i})$$

can be computed as $\beta_{X_i,j} = \rho_{ij} / \sqrt{\Sigma_{i,i} / \Sigma_{j,j}}$.

In practical applications, the linear regression parametrisation is more common. For the MARKS data, it works out as follows:

$$\begin{aligned}
\mathsf{ALG} &= 50.60 + \varepsilon_{\mathsf{ALG}} \sim N(0, 112.8) \\
\mathsf{ANL} &= -3.57 + 0.99\mathsf{ALG} + \varepsilon_{\mathsf{ANL}} \sim N(0, 110.25) \\
\mathsf{MECH} &= -12.36 + 0.54\mathsf{ALG} + 0.46\mathsf{VECT} + \varepsilon_{\mathsf{MECH}} \sim N(0, 195.2) \\
\mathsf{STAT} &= -11.19 + 0.76\mathsf{ALG} + 0.31\mathsf{ANL} + \varepsilon_{\mathsf{STAT}} \sim N(0, 158.8) \\
\mathsf{VECT} &= 12.41 + 0.75\mathsf{ALG} + \varepsilon_{\mathsf{VECT}} \sim N(0, 109.8)
\end{aligned}$$

Note that:

- parents only contribute main effects, no interactions;
- they only appear in their natural form, that is, no transforms.

Overall parameters of the $X_i \mid \Pi_{X_i}$: $11 + 5 = 16$

BNLEARN: Creating a Gaussian BN

```
marks.dag =
  model2network("[ALG][ANL|ALG][MECH|ALG:VECT][STAT|ALG:ANL][VECT|ALG]")

ALG.dist = list(coef = c("(Intercept)" = 50.60), sd = 10.62)
ANL.dist = list(coef = c("(Intercept)" = -3.57, ALG = 0.99), sd = 10.5)
MECH.dist =
  list(coef = c("(Intercept)" = -12.36, ALG = 0.54, VECT = 0.46), sd = 13.97)
STAT.dist =
  list(coef = c("(Intercept)" = -11.19, ALG = 0.76, ANL = 0.31), sd = 12.61)
VECT.dist = list(coef = c("(Intercept)" = 12.41, ALG = 0.75), sd = 10.48)

ldist = list(ALG = ALG.dist, ANL = ANL.dist, MECH = MECH.dist,
          STAT = STAT.dist, VECT = VECT.dist)
marks.bn = custom.fit(marks.dag, ldist)
```

Note that we specify the regression coefficients and the standard deviation of the residuals in keeping with the parameterisation used by R.

bnlearn: Local Linear Regressions

```
marks.bn[c("ALG", "STAT")]
```

```
$ALG

  Parameters of node ALG (Gaussian distribution)

Conditional density: ALG
Coefficients:
(Intercept)
       50.6
Standard deviation of the residuals: 10.6

$STAT

  Parameters of node STAT (Gaussian distribution)

Conditional density: STAT | ALG + ANL
Coefficients:
(Intercept)          ALG          ANL
     -11.19         0.76         0.31
Standard deviation of the residuals: 12.6
```

From Local to Global Distributions (I)

Reconstructing the global distribution from the local linear regression is also an interesting exercise. This is how `gbn2mvnorm()` does it:

1. Count the nodes and sort them in topological order (top to bottom in the DAG).

```
nodes = nodes(marks.bn)
nnodes = length(nodes)
ordered = node.ordering(marks.bn)
```

2. Compute the mean vector μ, sweeping the nodes in topological order.

```
mu = structure(rep(0, nnodes), names = nodes)
for (node in ordered) {
  pars = marks.bn[[node]]$parents
  coefs = marks.bn[[node]]$coefficients

  mu[node] = sum(c(1, mu[pars]) * coefs[c("(Intercept)", pars)])
}#FOR
mu
```

```
  ALG  ANL MECH STAT VECT
 50.6 46.5 38.1 41.7 50.4
```

From Local to Global Distributions (II)

3. Sweep the nodes again to construct an auxiliary matrix cumulating up the regression coefficients and the residual standard deviations in the lower triangular part.

```
chol = matrix(0, nrow = nnodes, ncol = nnodes,
         dimnames = list(ordered, ordered))
for (node in ordered) {
  pars = marks.bn[[node]]$parents
  coefs = marks.bn[[node]]$coefficients
  stderr = marks.bn[[node]]$sd

  chol[node, node] = stderr
  chol[node, ] = chol[node, ] + t(coefs[pars]) %*% chol[pars, ]
}
chol
```

```
         ALG   ANL  VECT MECH STAT
ALG   10.62  0.00  0.00    0  0.0
ANL   10.51 10.50  0.00    0  0.0
VECT   7.96  0.00 10.48    0  0.0
MECH   9.40  0.00  4.82   14  0.0
STAT 11.33  3.25  0.00    0 12.6
```

From Local to Global Distributions (III)

4. Compute the cross-product of `chol` to get the covariance matrix Σ.

```
sigma = (chol %*% t(chol))[nodes, nodes, drop = FALSE]
sigma
```

```
        ALG   ANL  MECH  STAT  VECT
ALG   112.8 111.7  99.8 120.3  84.6
ANL   111.7 220.8  98.8 153.3  83.7
MECH   99.8  98.8 306.7 106.5 125.4
STAT  120.3 153.3 106.5 298.0  90.2
VECT   84.6  83.7 125.4  90.2 173.3
```

For comparison, the raw value from the original MARKS data set:

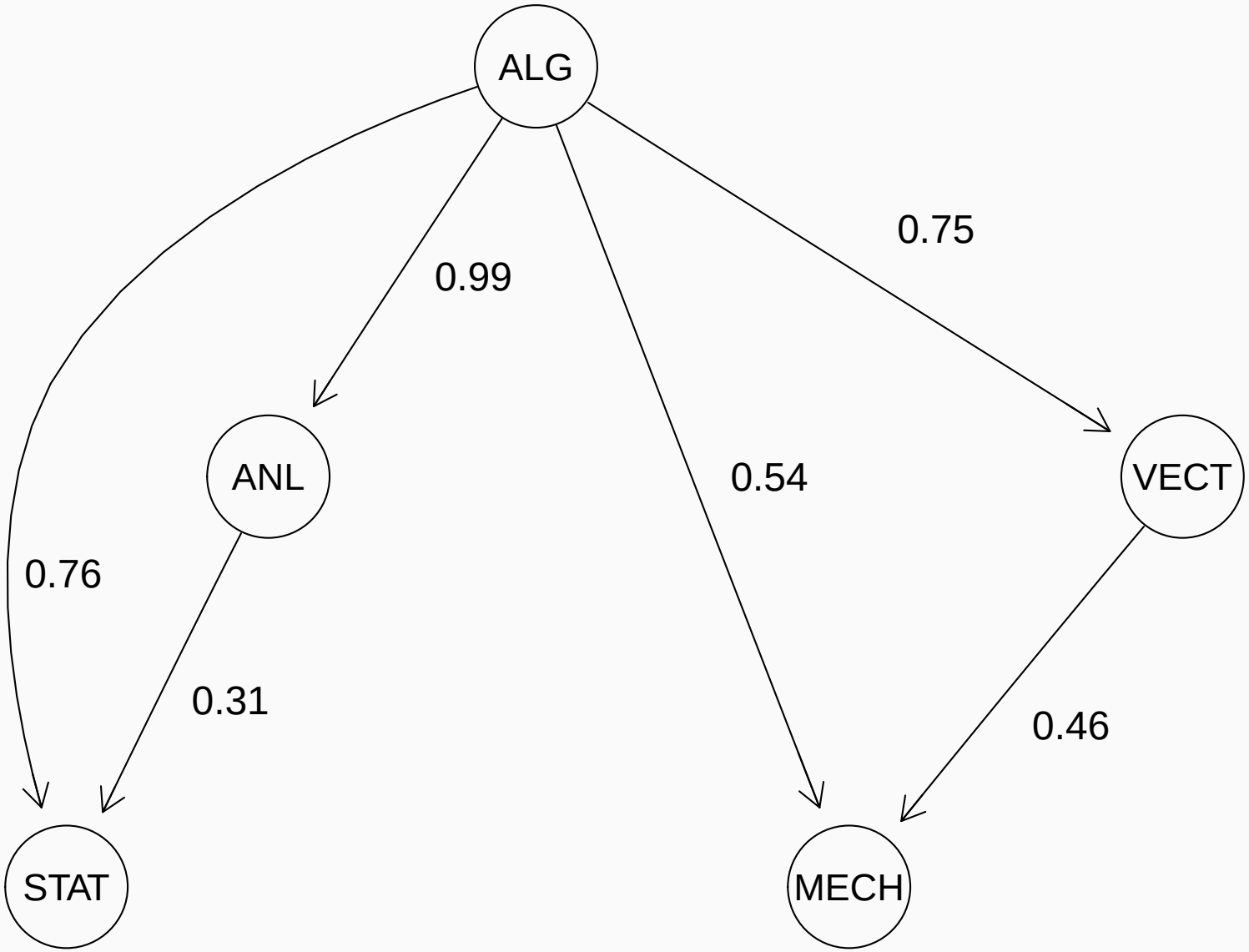
```
colMeans(marks)[nodes]
```

```
  ALG   ANL MECH STAT VECT
 50.6  46.7 39.0 42.3 50.6
```

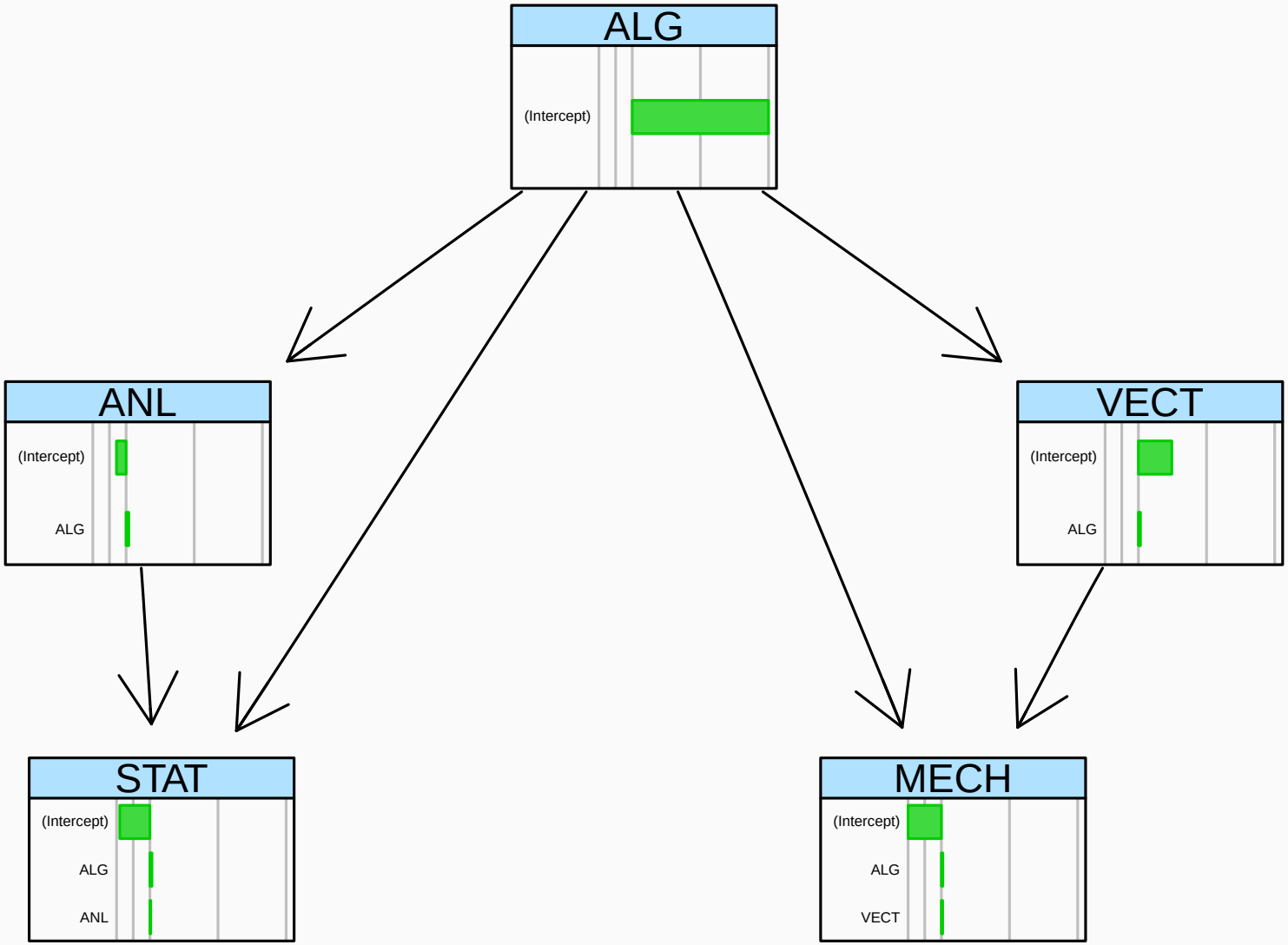
```
cov(marks)[nodes, nodes]
```

```
        ALG   ANL MECH STAT  VECT
ALG   112.9 112.1  102  122  85.2
ANL   112.1 220.4  106  156  94.7
MECH  101.6 106.3  306  117 127.2
STAT  121.9 155.5  117  298  99.0
VECT   85.2  94.7  127   99 172.8
```

Gaussian Bayesian Networks and Wright's Path Analysis



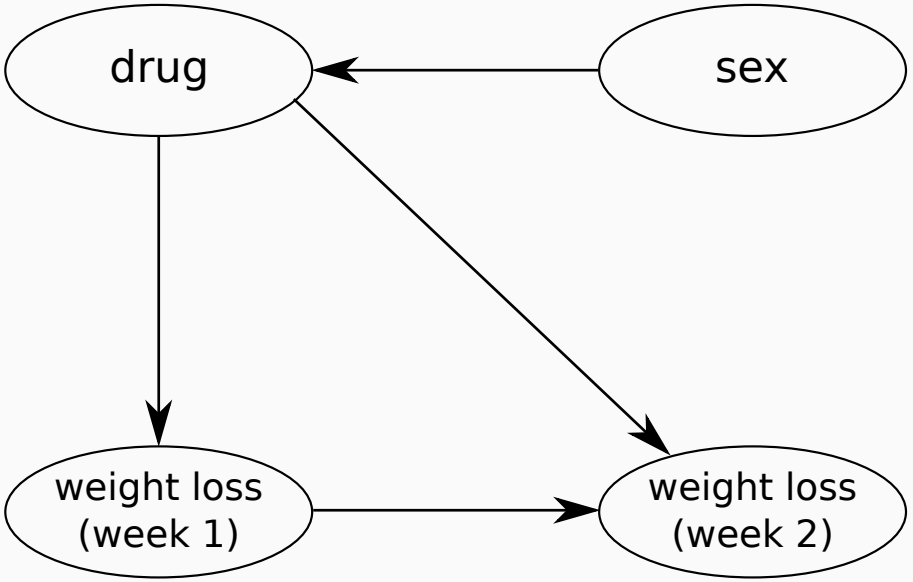
Canonical Plot of a GBN



Conditional Linear Gaussian BNs

CGBNs contain both discrete and continuous nodes, and combine DBNs and GBNs as follows to obtain a mixture-of-Gaussians distribution:

- continuous nodes cannot be parents of discrete nodes;
- the local distribution of each discrete node is a CPT;
- the local distribution of each continuous node is a set of linear regression models, one for each configurations of the discrete parents, with the continuous parents acting as regressors.



A classic example is the RATS' WEIGHTS network from Edwards (1995), which describes weight loss in a drug trial on rats.

The resulting local distribution for the first weight loss for drugs "D1", "D2" and "D3" is:

$$\begin{aligned}
\texttt{WL1}_{\texttt{D1}} &= 7 + \varepsilon_{\texttt{WL1}\,|\,\texttt{D1}} \sim N(0, 2.5) \\
\texttt{WL1}_{\texttt{D2}} &= 7.50 + \varepsilon_{\texttt{WL1}\,|\,\texttt{D2}} \sim N(0, 2) \\
\texttt{WL1}_{\texttt{D3}} &= 14.75 + \varepsilon_{\texttt{WL1}\,|\,\texttt{D3}} \sim N(0, 11)
\end{aligned}$$

with just the intercepts since WL1 has no continuous parents. The local distribution for the second weight loss is:

$$\begin{aligned}
\texttt{WL2}_{\texttt{D1}} &= 1.02 + 0.89\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D1}} \sim N(0, 3.2) \\
\texttt{WL2}_{\texttt{D2}} &= -1.68 + 1.35\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D2}} \sim N(0, 4) \\
\texttt{WL2}_{\texttt{D3}} &= -1.83 + 0.82\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D3}} \sim N(0, 1.9)
\end{aligned}$$

and contains a regression coefficient for WL1.

The Global Distribution (I)

The discrete nodes define the components of a mixture, each component is the joint multivariate distribution of the continuous nodes.

Drug has states "D1", "D2" and "D3". Sex has states "M" and "F". So the mixture has 6 components:

$$\Big\{\{\texttt{"D1"},\texttt{"M"}\},\{\texttt{"D1"},\texttt{"F"}\},\{\texttt{"D2"},\texttt{"M"}\},\{\texttt{"D2"},\texttt{"F"}\},\{\texttt{"D3"},\texttt{"M"}\},\{\texttt{"D3"},\texttt{"F"}\}\Big\}$$

The first component, for which Drug is "D1" and sex is "M", identifies the local distributions of weight loss as:

$$\begin{aligned}\texttt{WL1}_{\texttt{D1}} &= 7 + \varepsilon_{\texttt{WL1}\,|\,\texttt{D1}} \sim N(0, 2.5)\\ \texttt{WL2}_{\texttt{D1}} &= 1.02 + 0.89\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D1}} \sim N(0, 3.2)\end{aligned}$$

Together, they can be thought of as a GBN and therefore we can write them as a multivariate normal distribution as we did earlier.

The Global Distribution (II)

The component for {"D1", "F"} is identical to that for {"D1", "M"} because weight loss is independent from sex given drug.

Along the same lines, the components for both {"D2", "M"} and {"D2", "F"} identify the local distributions of weight loss as:

$$\begin{aligned}\texttt{WL1}_{\texttt{D2}} &= 7.50 + \varepsilon_{\texttt{WL1}\,|\,\texttt{D2}} \sim N(0, 2)\\ \texttt{WL2}_{\texttt{D2}} &= -1.68 + 1.35\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D2}} \sim N(0, 4)\end{aligned}$$

For {"D3", "M"} and {"D3", "F"}:

$$\begin{aligned}\texttt{WL1}_{\texttt{D3}} &= 14.75 + \varepsilon_{\texttt{WL1}\,|\,\texttt{D3}} \sim N(0, 11)\\ \texttt{WL2}_{\texttt{D3}} &= -1.83 + 0.82\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D3}} \sim N(0, 1.9)\end{aligned}$$

In total, we have a mixture model with six components, each with a bivariate normal distribution.

BNLEARN: Creating a Conditional Linear Gaussian BN

```
rats.dag = model2network("[SEX][DRUG|SEX][WL1|DRUG][WL2|WL1:DRUG]")
SEX.lv = c("M", "F")
DRUG.lv = c("D1", "D2", "D3")

SEX.prob = array(c(0.5, 0.5), dim = 2, dimnames = list(SEX = SEX.lv))
DRUG.prob = array(c(0.3333, 0.3333, 0.3333, 0.3333, 0.3333, 0.3333),
                  dim = c(3, 2), dimnames = list(DRUG = DRUG.lv, SEX = SEX.lv))
WL1.coef = matrix(c(7, 7.50, 14.75), nrow = 1, ncol = 3,
                  dimnames = list("(Intercept)", NULL))
WL1.dist = list(coef = WL1.coef, sd = c(1.58, 0.447, 3.31))
WL2.coef = matrix(c(1.02, 0.89, -1.68, 1.35, -1.83, 0.82), nrow = 2, ncol = 3,
                  dimnames = list(c("(Intercept)", "WL1")))
WL2.dist = list(coef = WL2.coef, sd = c(1.78, 2, 1.37))

ldist = list(SEX = SEX.prob, DRUG = DRUG.prob, WL1 = WL1.dist, WL2 = WL2.dist)
rats.bn = custom.fit(rats.dag, ldist)
```

The regression coefficients are stored in a matrix with one conditional regression in each column: each column corresponds to one configuration of the discrete parents and each row to one of the continuous parents.

BNLEARN: Mixtures of Linear Regressions

```
bn$WL2
```

```
  Parameters of node WL2 (conditional Gaussian distribution)

Conditional density: WL2 | DRUG + WL1
Coefficients:
                  0      1      2
(Intercept)    1.02  -1.68  -1.83
WL1            0.89   1.35   0.82
Standard deviation of the residuals:
   0     1     2
1.78  2.00  1.37
Discrete parents' configurations:
   DRUG
0    D1
1    D2
2    D3
```

The local distribution of each continuous node contains a set of regressions, all with the same coefficients. Consider WL2: we can write the intercept and each coefficient as the deviation from the respective means across the drugs.

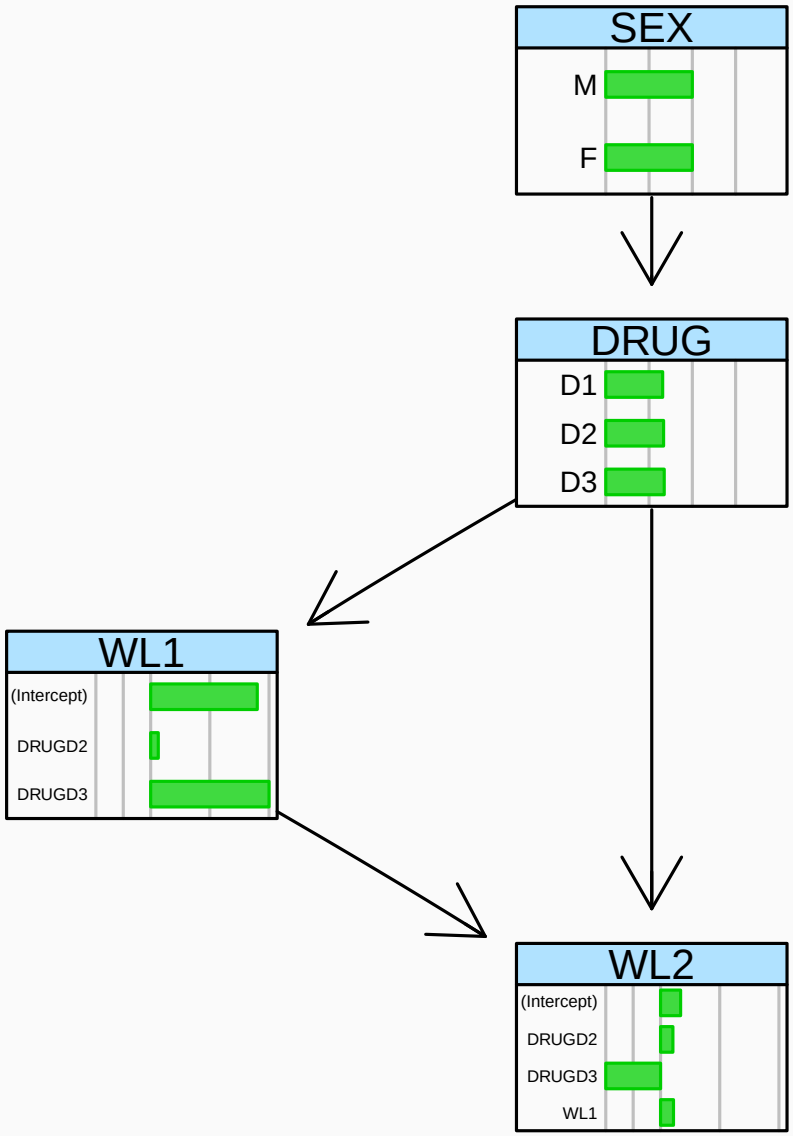
$$\begin{aligned}
\texttt{WL2}_{\texttt{D1}} &= (1.85 - 0.83) + (-0.13 + 1.02)\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D1}} \\
\texttt{WL2}_{\texttt{D2}} &= (-0.85 - 0.83) + (0.33 + 1.02)\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D2}} \\
\texttt{WL2}_{\texttt{D3}} &= (-1.00 - 0.83) + (-0.20 + 1.02)\texttt{WL1} + \varepsilon_{\texttt{WL2}\,|\,\texttt{D3}}
\end{aligned}$$

This representation looks much like a mixed-effects models with a fixed intercept (0.83), a fixed slope (1.02), a random intercept (one for each drug) and a random slope (one for each drug).

Limitations of These Probability Distributions

- No real-world data set follows a multivariate Gaussian distribution: even if the marginal distributions are normal, not all dependence relationships are linear.
- Computing partial correlations can be problematic because of singularities.
- Parametric assumptions for mixed data have strong limitations: they impose constraints on which arcs may be present in the DAG.
- Discretisation is a common solution to the above problems, but it may discard useful information and it is tricky to get right (choosing a set of intervals such that the dependence relationships involving the original variable are preserved). On the other hand, dependencies are no longer required to be linear.
- Ordinal variables are treated as categorical, again losing information.

Canonical Plot of a CGBN



If the structure of the model is known, the problem of estimating the parameters of the global distribution can be solved by estimating the parameters of each local distribution, one at a time.

Common choices are:

- Maximum likelihood estimators: just the usual empirical estimators. Often described as either maximum entropy or minimum divergence estimators in information-theoretic literature.
- Bayesian posterior estimators: based on conjugate priors to keep computations fast, simple and in closed form.
- Shrinkage estimators: regularised estimators based either on James-Stein or Bayesian shrinkage results.

Maximum Likelihood and Maximum Entropy Estimators

The classic estimators for (conditional) probabilities and (partial) correlations / regression coefficients have known limitations:

- maximum likelihood estimates are unstable in most multivariate problems, both discrete and continuous;
- for the multivariate Gaussian distribution, James & Stein proved in the 1950s that the maximum likelihood estimator for the mean is not admissible in 3+ dimensions;
- partial correlations are often ill-behaved because of that, even with Moore-Penrose pseudo-inverses;
- maximum likelihood estimates are non-smooth and create problems when using the graphical model for inference.

Maximum a Posteriori Bayesian Estimators

Bayesian posterior estimates are the sensible choice for parameter estimation according to Koller's & Friedman's tome on graphical models. Choices for the priors are limited (for computational reasons) to conjugate distributions, namely:

- the Dirichlet for discrete models,

$$Dir(\alpha_{k\,|\,\Pi_{X_i}=\pi}) \quad \stackrel{data}{\longrightarrow} \quad Dir(\alpha_{k\,|\,\Pi_{X_i}=\pi} + n_{k\,|\,\Pi_{X_i}=\pi})$$

  meaning that $\hat{p}_{k\,|\,\Pi_{X_i}=\pi} = \alpha_{k\,|\,\Pi_{X_i}=\pi} / \sum_{\pi} \alpha_{k\,|\,\Pi_{X_i}=\pi}$;
- the Inverse Wishart for Gaussian models,

$$IW(\Psi, m) \quad \stackrel{data}{\longrightarrow} \quad IW(\Psi + n\Sigma, m + n).$$

In both cases, the only free parameter with a non-informative prior is the equivalent or imaginary sample size, which gives the relative weight of the prior compared to the observed sample.

Bayesian LASSO and Ridge Regression

Gaussian graphical models, being closely related with linear regression, have also used ridge regression (L_2 regularisation) and LASSO (L_1 regularisation) in their Bayesian capacity.

LASSO corresponds to a Laplace prior on the regression coefficients,

$$\beta_k \,|\, \sigma^2 \sim Laplace(0, \sigma^2).$$

Ridge Regression corresponds to a Gaussian prior,

$$\beta_k \,|\, \sigma^2 \sim N(0, \sigma^2).$$

In both cases, tuning the σ^2 parameter is crucial, as it takes the role of the λ regularisation parameter found in the original frequentist definitions of these methods. Furthermore, excessive regularisation might lead to zero coefficients that would make a node independent of its parents.

Shrinkage estimation is based on results from James & Stein on the estimation of the mean of a multivariate Gaussian distribution, and takes the form

$$\tilde{\theta} = \lambda t + (1 - \lambda)\hat{\theta} \qquad \lambda \in [0, 1]$$

where the optimal λ (with respect to squared loss) can be estimated in closed form as

$$\lambda^* = \min \left(\frac{\sum_k \mathrm{VAR}(\hat{\theta}_k) - \mathrm{COV}(\hat{\theta}_k, t_k) + \mathrm{Bias}(\hat{\theta}_k)\,\mathrm{E}(\hat{\theta}_k - t_k)}{\sum_k (\hat{\theta}_k - t_k)^2}, 1 \right).$$

The James-Stein estimator $\tilde{\theta}$ dominates the maximum likelihood estimator $\hat{\theta}$ and converges to the latter as the sample size grows. It can be interpreted as an empirical Bayes estimator.

For discrete data, conditional probabilities $\pi_{ik\,|\,j}$ are estimated as

$$\tilde{\pi}_{ikj} = \lambda^* t_{ikj} + (1-\lambda^*)\hat{\pi}_{ikj}, \quad \lambda^* = \min\left(\frac{1-\sum_k \hat{\pi}^2_{ikj}}{(n-1)\sum_k (t_{ikj}-\hat{p}_{ikj})^2}, 1\right),$$

where t is the uniform (discrete) distribution.

For continuous data, correlations end up being estimated from the shrunk covariance matrix $\widetilde{\Sigma}$

$$\tilde{\sigma}_{ii} = \hat{\sigma}_{ii}, \quad \tilde{\sigma}_{ij} = (1-\lambda^*)\hat{\sigma}_{ij}, \quad \lambda^* = \min\left(\frac{\sum_{i\neq j} \mathrm{VAR}(\hat{\sigma}_{ij})}{\sum_{i\neq j} \hat{\sigma}^2_{ij}}, 1\right)$$

where t is $\mathrm{diag}(\widehat{\Sigma})$. $\widetilde{\Sigma}$ is guaranteed to have full rank, so it can be safely inverted to get partial correlations.

Parameter learning is implemented in `bn.fit()` and defaults to `method = "mle"`; for discrete data we can also use Bayesian posterior estimation with `method = "bayes"` with an imaginary sample size `iss`.

```
fitted = bn.fit(asia.dag, asia, method = "mle")
coef(fitted$X)
```

```
      E
X          no     yes
  no  0.95659 0.00541
  yes 0.04341 0.99459
```

```
fitted = bn.fit(asia.dag, asia, method = "bayes", iss = 20)
coef(fitted$X)
```

```
      E
X          no    yes
  no  0.9556 0.0184
  yes 0.0444 0.9816
```

bnlearn implements only `method = "mle-g"` for GBNs, but we can use `penalized()` to replace parameter estimates with ridge, LASSO, or elastic net estimates.

```
library(penalized)
fitted = bn.fit(marks.dag, marks, method = "mle-g")
coef(fitted$STAT)
```

```
  (Intercept)          ALG          ANL
      -11.192        0.765        0.316
```

```
fitted$STAT = penalized(response = marks[, "STAT"],
                penalized = marks[, parents(fitted, "STAT")],
                lambda2 = 100, model = "linear", trace = FALSE)
coef(fitted$STAT)
```

```
  (Intercept)          ALG          ANL
      -10.788        0.753        0.321
```

We can also fit the parameters directly using a DAG and any other regression functions, and collecting their coefficients in a BN with `custom.fit()`.

Same for CGBNs.

```
library(lme4)
data(rats, library = "gRbase")
rats[, c("W1", "W2")] = list(as.numeric(rats$W1), as.numeric(rats$W2))
names(rats) = c("SEX", "DRUG", "WL1", "WL2")
fitted = bn.fit(rats.dag, data = rats, method = "mle-cg")
coef(fitted$WL2)
```

```
                0     1      2
(Intercept) 1.028 -1.68 -1.835
WL1         0.889  1.36  0.819
```

```
ldist = list(coef = array(0, dim = c(2, 3),
          dimnames = list(c("(Intercept)", "WL1"), NULL)), sd = rep(0, 3))
model = lmer(WL2 ~ WL1 + (1 | DRUG), data = rats)
for (i in seq(ncol(ldist$coef)))
  ldist$coef[, i] = fixef(model) + ranef(model)[["DRUG"]][i, ]
for (i in seq(length(ldist$sd)))
  ldist$sd[i] = sd(resid(model)[rats[, "DRUG"] == levels(rats[, "DRUG"])[i]])
fitted$WL2 = ldist
coef(fitted$WL2)
```

```
                0    1      2
(Intercept) 1.57 2.36 -1.42
WL1         1.54 2.32 -1.45
```

The definition of a BN just says

$$\mathrm{P}(\mathbf{X}, \Theta) = \prod_{i=1}^{N} \mathrm{P}(X_i \mid \Pi_{X_i}; \Theta_{X_i}) \quad \text{where} \quad \Pi_{X_i} = \{\text{parents of } X_i\}.$$

We are free to choose our $\mathrm{P}(X_i \mid \Pi_{X_i}; \Theta_{X_i})$, treating any hierarchical Bayesian model as a BN and using it as such. However, we should include nodes only for the variables in the model, not their parameters, to be able to interpret it in the same way as a classical BN.

What we gain:

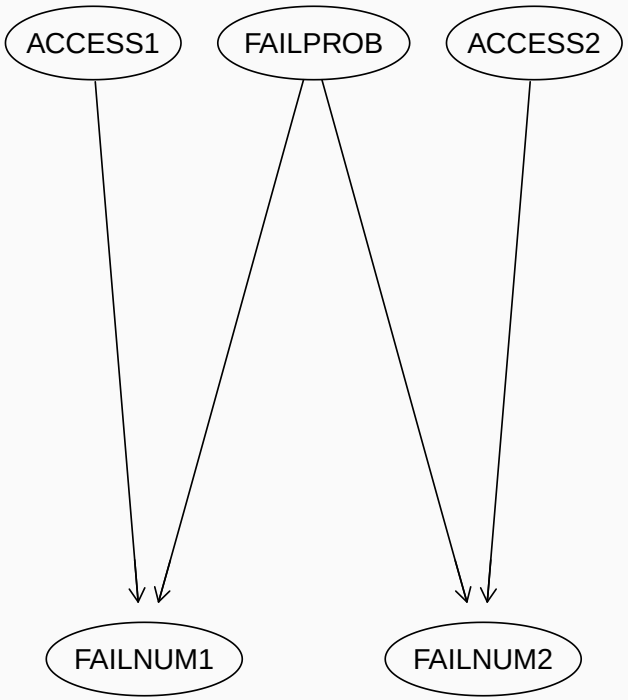
- Freedom! Power!
- The ability to use Bayesian modelling software such as Stan.

What we lose:

- Closed-form results for density functions, parameter estimates, scores, tests, etc.
- The ability to use **bnlearn**.

A General BN

A simple example is the RELIABILITY TESTING model for two-phase testing (somewhat amended here) by Fenton & Neil, assessing the distribution of the number of failures based on the number of accesses and the system's reliability.



$$\texttt{FAILPROB} \sim Beta\,(2,5)$$
$$\texttt{ACCESS1} \sim Pois\,(20)$$
$$\texttt{ACCESS2} \sim Pois\,(400)$$
$$\texttt{FAILNUM1} \sim Bi\,(\texttt{ACCESS1}, \texttt{FAILPROB})$$
$$\texttt{FAILNUM2} \sim Bi\,(\texttt{ACCESS2}, \texttt{FAILPROB})$$

RSTAN: Building a General BN as a Hierarchical Model (I)

This is the Stan model specification to generate observations from the BN, given its parameters. We store that as a string in a variable `stancode` to use Stan through R via **rstan**.

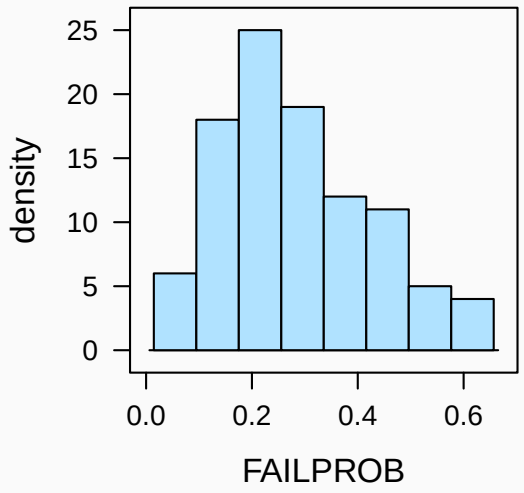
```
data {
  vector[2] Fp;  // shape parameters for the beta distribution.
  real A1p;      // expected accesses per second, phase 1
  real A2p;      // expected accesses per second, phase 2.
}
generated quantities {
  real FAILPROB;
  int ACCESS1;
  int ACCESS2;
  int FAILNUM1;
  int FAILNUM2;

  FAILPROB = beta_rng(Fp[1], Fp[2]);
  ACCESS1 = poisson_rng(A1p);
  ACCESS2 = poisson_rng(A2p);
  FAILNUM1 = binomial_rng(ACCESS1, FAILPROB);
  FAILNUM2 = binomial_rng(ACCESS2, FAILPROB);
}
```

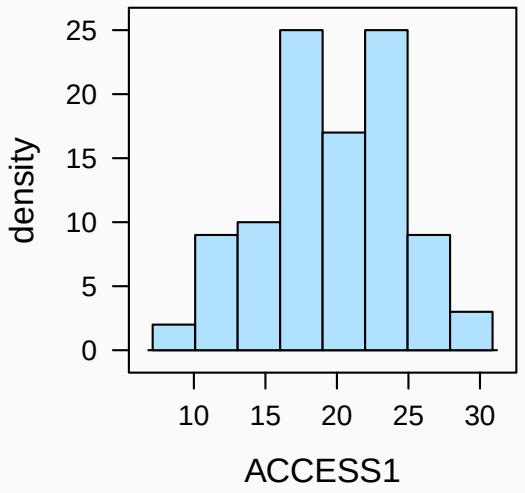
rstan: Building a General BN as a Hierarchical Model (II)

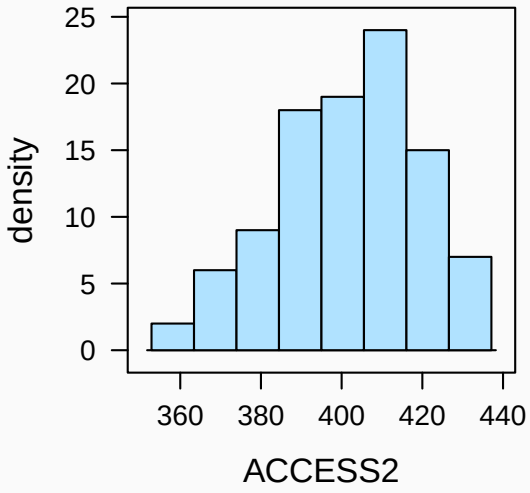
```
library(rstan)
reliability.bn = stan_model(model_code = stancode)
params = list(
  Fp = c(2, 5),
  A1p = 20,
  A2p = 400
)
data = sampling(reliability.bn, algorithm = "Fixed_param", data = params,
        thin = 10, iter = 500, seed = 42)
nodes = c("FAILPROB", "ACCESS1", "ACCESS2", "FAILNUM1", "FAILNUM2")
reliability = as.data.frame(extract(data)[nodes])
```

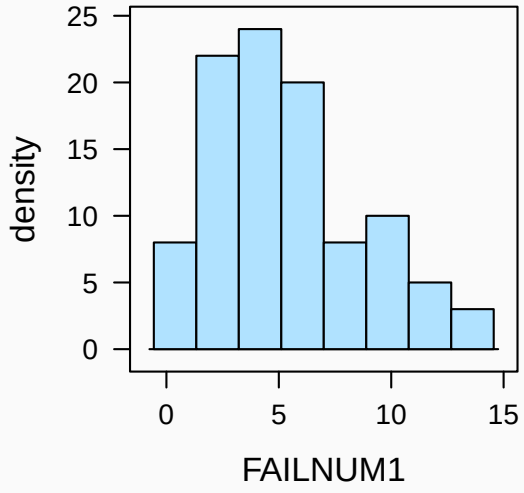
Unlike **bnlearn**'s `bn.fit` objects, **rstan**'s models are not accessible programmatically from R because they live in the C++ code of the Stan library.

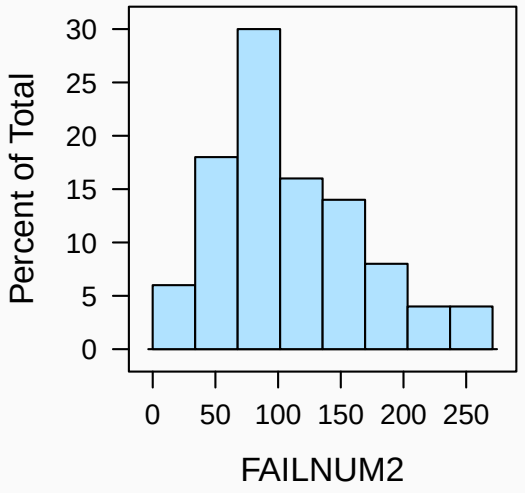
Some Plots of the Empirical Marginal Distributions











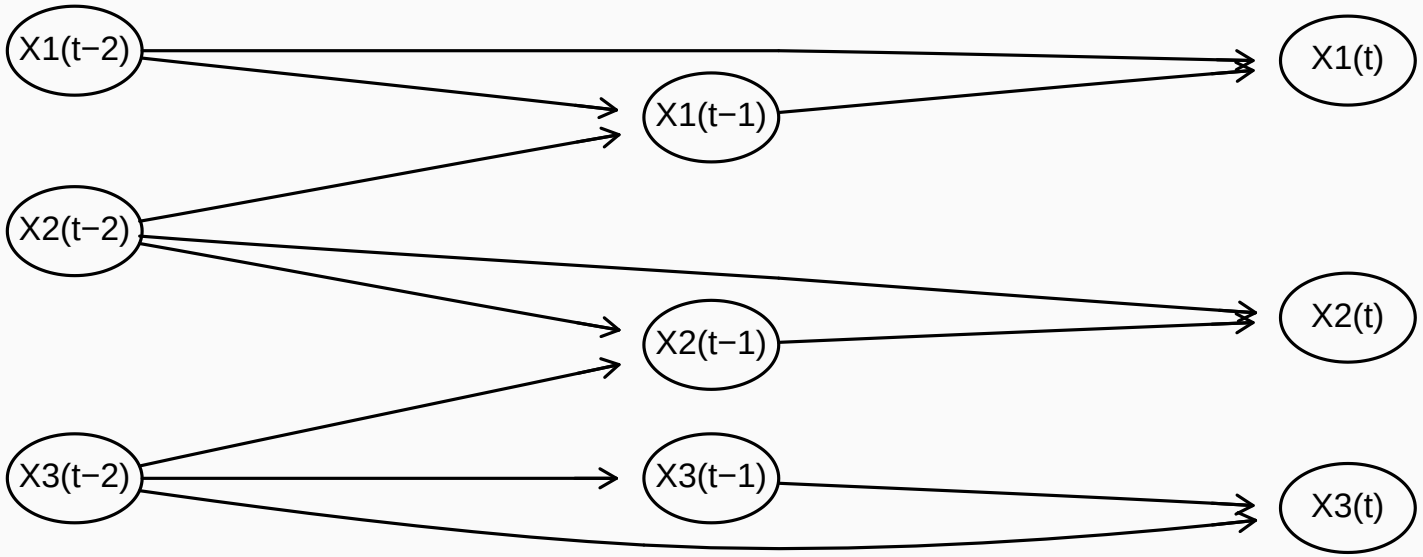
Vector Auto-Regressive Processes

Multivariate time series can be modelled as vector auto-regressive (VAR) processes. A $\mathrm{VAR}(\mathbf{X}, p)$ of order p has the form

$$\mathbf{X}(t) = A_1\mathbf{X}(t-1) + ... + A_p\mathbf{X}(t-p) + B + \varepsilon_{\mathbf{X}}, (t)$$

where $\mathbf{X}(t)$ are the variables measured at time t.

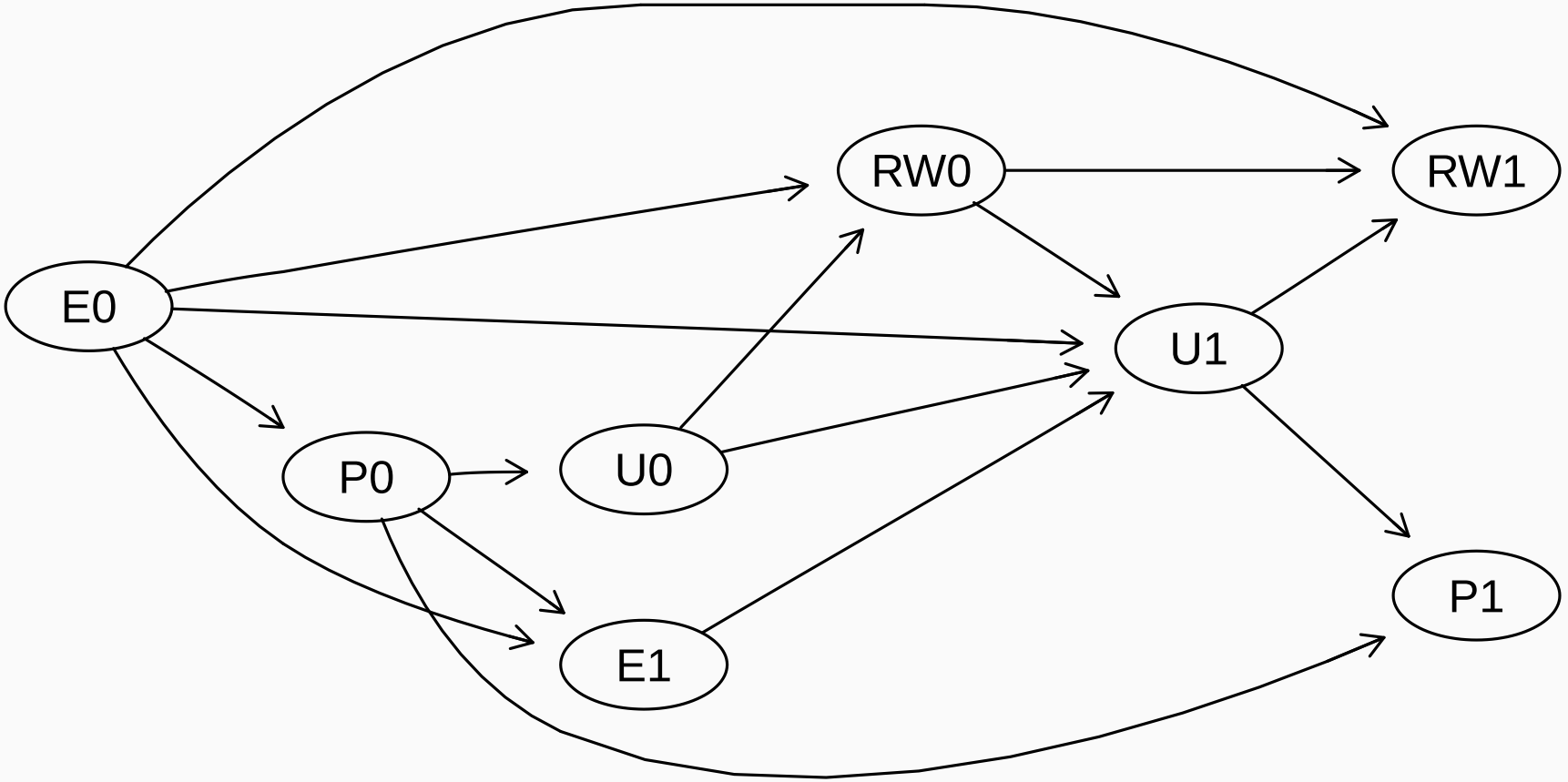
In graphical form, for instance ($p = 2$):



Does it remind you of something?

Dynamic BNs

The classic CANADA macroeconomics time series, studied by Lütkepohl and Pfaff. E is employment, P is productivity, RW is the real wage and U is the unemployment rate.



The Graphical Structure

Any stochastic process in discrete time has a similar graphical representation and can be treated as a BN. In such a dynamic BN:

- All variables $\mathbf{X}$ become separate nodes in each time point.
- Arcs can only go forward in time: the graph is still a DAG.
- We may allow instantaneous arcs between nodes in the same time point, but their interpretation is often not straightforward.
- We can model feedback loops between variables.

We typically assume that:

- arcs can only come from nodes in the previous time point (so, the time series is of order $p = 1$).
- no instantaneous arcs, unless time points are averages over periods.
- the time series is homogeneous and stationary.

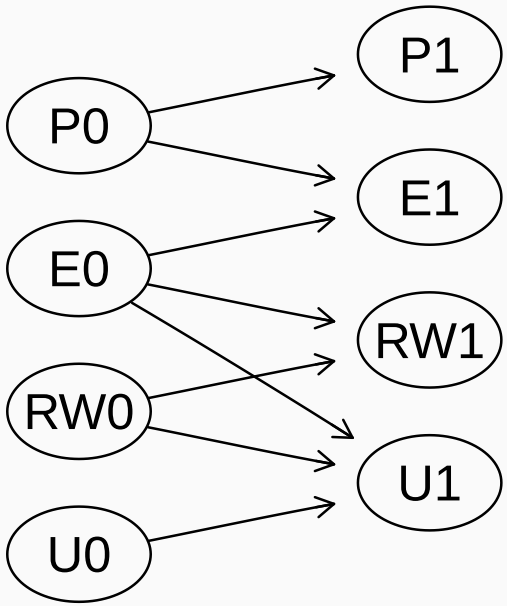
We do not assume any particular distribution for the nodes.

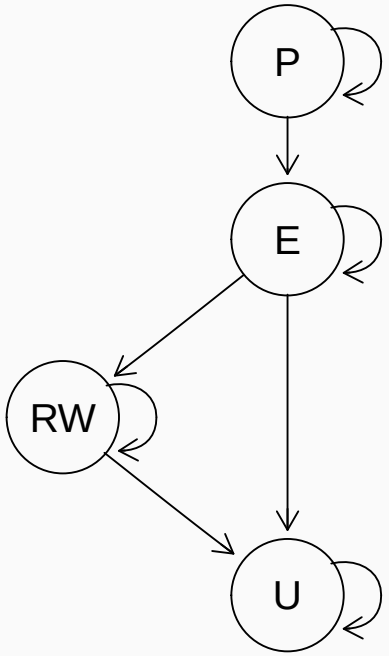
Rolled-Up and Unrolled Graphs for a Dynamic BN

With these assumptions, we can completely represent a dBN with:

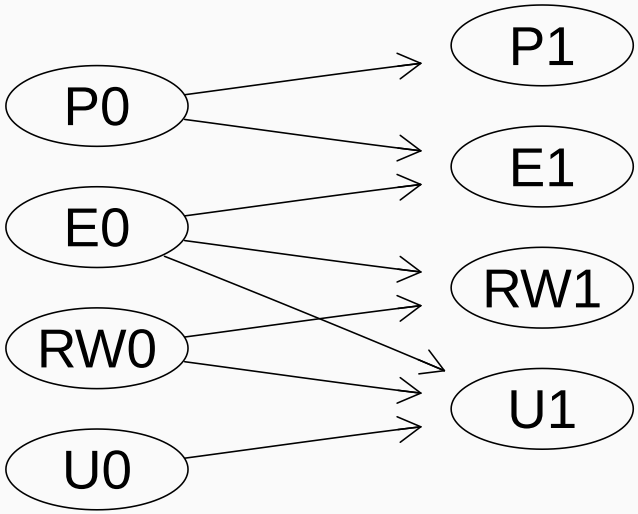
- a 2-time BN representing the transition between two consecutive time points;
- the marginal distribution of each variable in $\mathbf{X}$, which models the dBN at time $t = 0$.

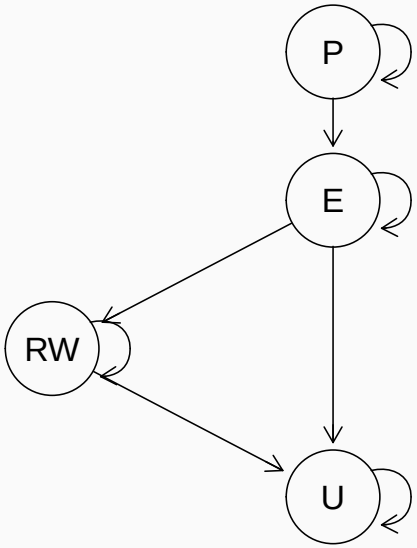
The 2-time BN has two representations: unrolled and rolled-up.

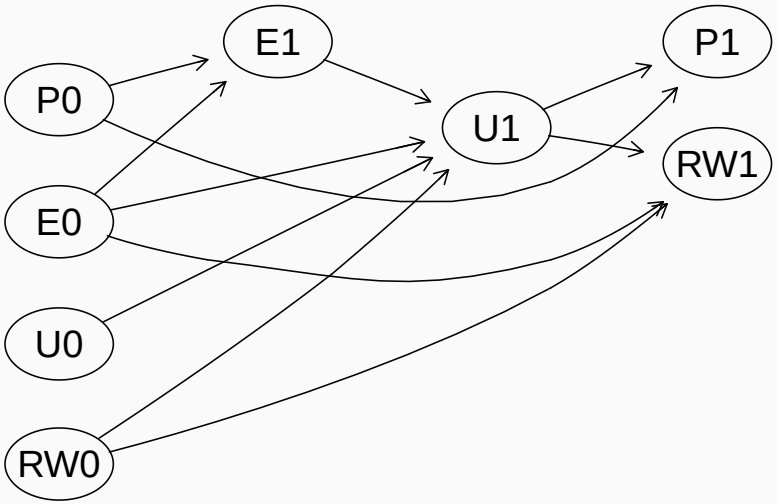


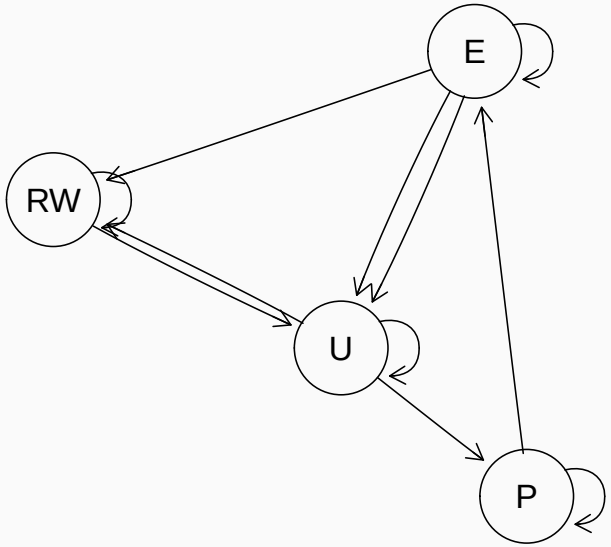


2-Time BNs With and Without Instantaneous Arcs







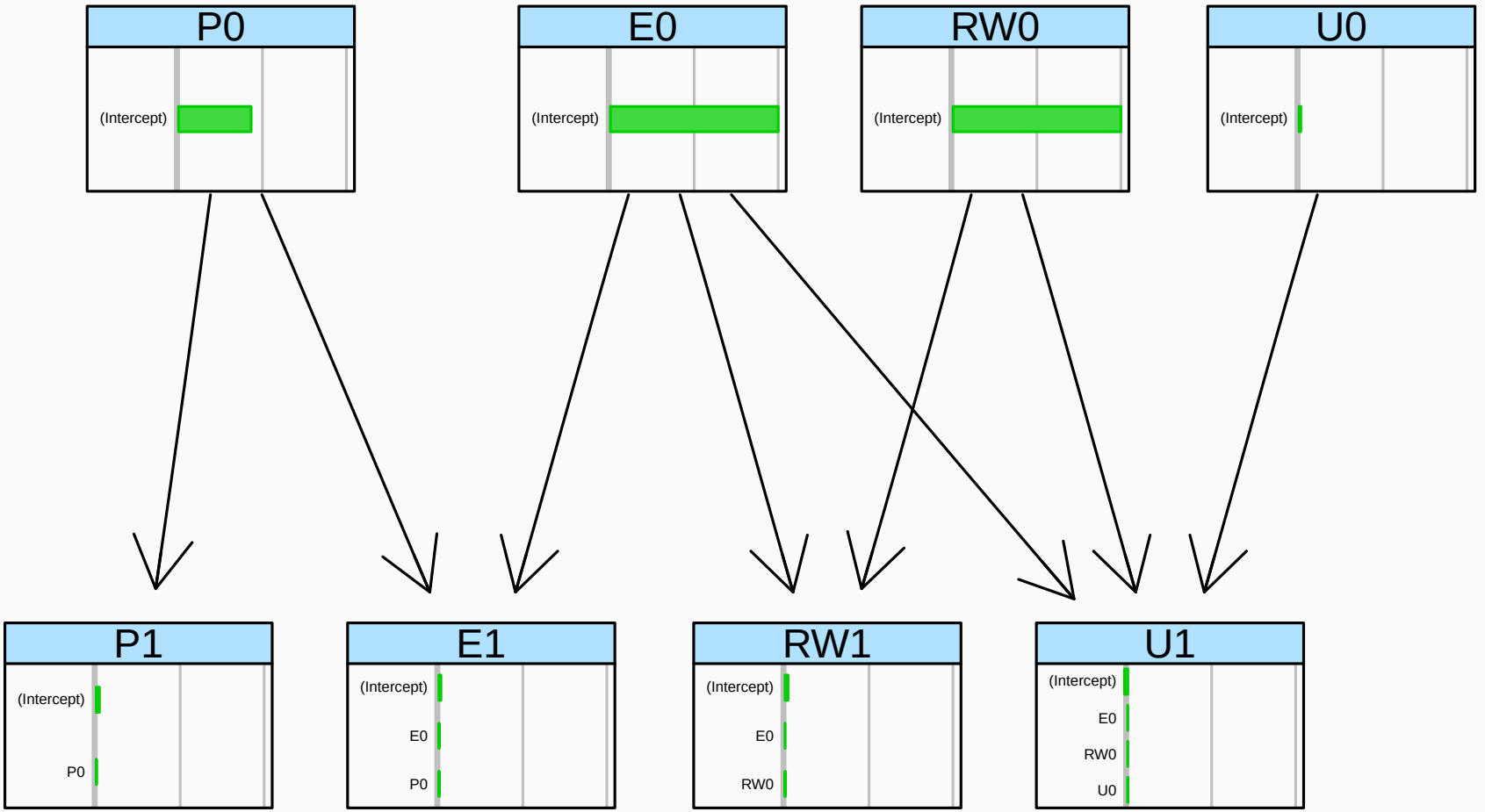


bnlearn: Creating a Dynamic BN for a VAR

```
canada.dag =
  model2network("[P0][E0][U0][RW0][P1|P0][E1|E0:P0][U1|U0:E0:RW0][RW1|RW0:E0]")
P0.dist = list(coef = c("(Intercept)" = 408), sd = 4.23)
E0.dist = list(coef = c("(Intercept)" = 944), sd = 9.07)
U0.dist = list(coef = c("(Intercept)" = 9.34), sd = 1.6)
RW0.dist = list(coef = c("(Intercept)" = 944), sd = 9.07)
P1.dist = list(coef = c("(Intercept)" = 17.51, P0 = 0.96), sd = 0.70)
E1.dist = list(coef = c("(Intercept)" = 10.89, E0 = 1.04, P0 = 1.04), sd = 0.52)
U1.dist = list(coef = c("(Intercept)" = -17.85, E0 = 0.02, RW0 = -0.004,
                        U0 = 1.004), sd = 0.43)
RW1.dist = list(coef = c("(Intercept)" = 16.45, E0 = -0.04, RW0 = 1.03),
                sd = 0.88)
ldist = list(P0 = P0.dist, E0 = E0.dist, U0 = U0.dist, RW0 = RW0.dist,
             P1 = P1.dist, E1 = E1.dist, U1 = U1.dist, RW1 = RW1.dist)
canada.bn = custom.fit(canada.dag, ldist)
```

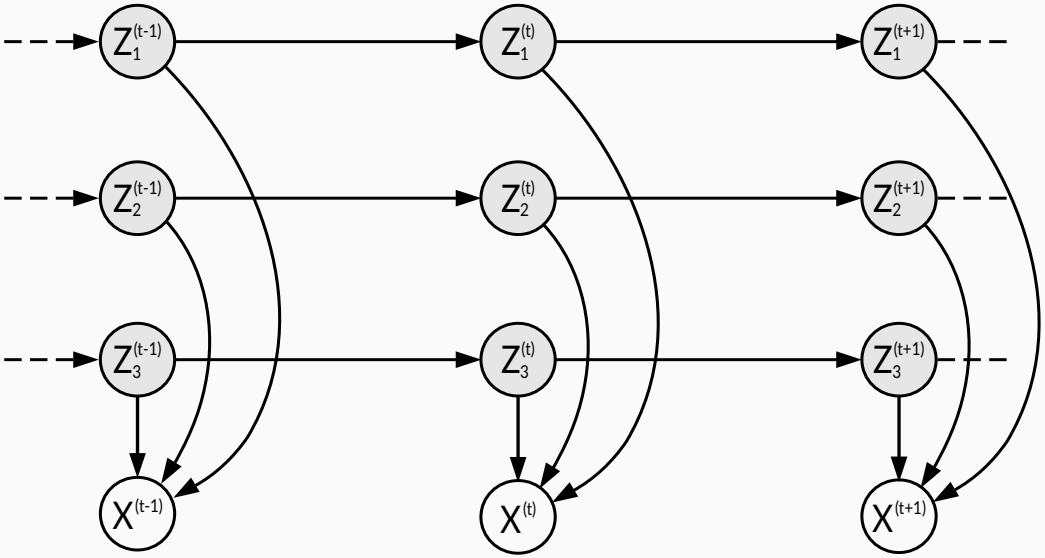
Code-wise, it looks much like creating a GBN. If the nodes were discrete, the code would look like that for creating a DBN. With mixed node types, the code would look like that for creating CGBNs or general BNs.

Canonical Plots Work for Dynamic BNs as Well

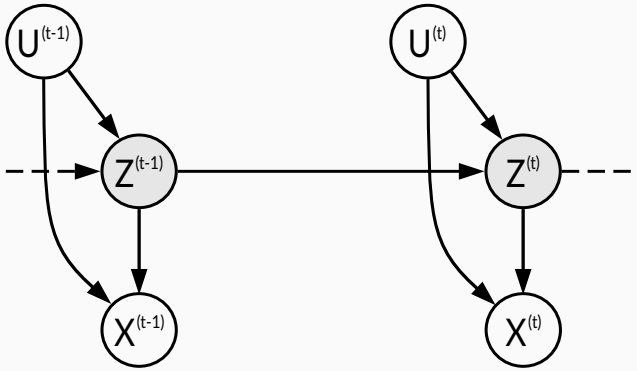


Many Models are Dynamic BNs

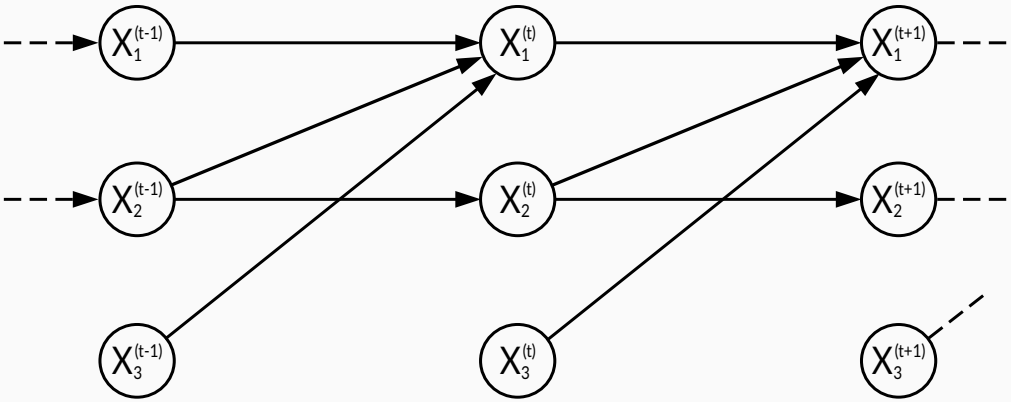
Factorial HMM



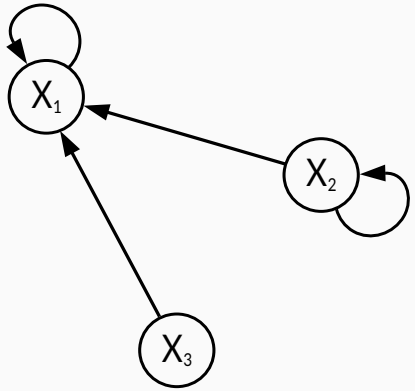
Kalman Filter



Unrolled VAR



Rolled VAR



Dynamic BN: Hidden Markov Models

Hidden Markov models (HMMs) are one of the most widespread approaches to model phenomena with hidden state, that is, in which the behaviour of the observed variables $\mathbf{X}$ depends on that of one or more discrete latent variables $\mathbf{Z}$ as well as on other variables in $\mathbf{X}$.

In dBN terms, an HMM model with M latent variables can be written as

$$\mathrm{P}\left(\mathbf{X}^{(t)} \mid \mathbf{X}^{(t-1)}, \mathbf{Z}^{(t)}\right) = \prod_{i=1}^{N} \mathrm{P}\left(X_i^{(t)} \mid \Pi_{X_i^{(t)}}, \mathbf{Z}^{(t)}\right)$$

$$\mathrm{P}\left(\mathbf{Z}^{(t)}\right) = \prod_{j=1}^{M} \mathrm{P}\left(Z_j^{(t)} \mid \Pi_{Z_j^{(t)}}\right),$$

with the restriction that the parents of $Z_j^{(t)}$ can only be other latent variables. In the vast majority of the literature all variables are assumed to be discrete. Depending on the choice of $\Pi_{Z_j^{(t)}}$, we can obtain various HMM variants such as hierarchical HMMs and factorial HMMs.

Dynamic BN: Vector Auto-Regressive Models

Vector auto-regressive models (VARs) are a straightforward multivariate extension of univariate auto-regressive time series for continuous variables.

VARs are defined as

$$\mathbf{X}^{(t)} = A_1\mathbf{X}^{(t-1)} + ... + A_L\mathbf{X}^{(t-L)} + \varepsilon_t, \qquad \varepsilon_t \sim N(0, \Sigma),$$

for some fixed Markov order L. We can rewrite that as

$$\mathbf{X}^{(t)} \mid \mathbf{X}^{(t-1)}, ..., \mathbf{X}^{(t-L)} \sim N\left(A_1\mathbf{X}^{(t-1)} + ... + A_L\mathbf{X}^{(t-L)}, \varepsilon_t\right)$$

and then restrict the parents of each $X_i^{(t)}$ to those for which the corresponding regression coefficients in $A_1, ..., A_L$ are different from zero using the one-to-one correspondence between regression coefficients and partial correlations. Formally, $X_j^{(t-l)} \in \Pi_{X_i^{(t)}}$ if and only if $A_l[i, j] \neq 0$, which makes it possible to write a VAR as a Gaussian dBN.

Dynamic BNs: Kalman Filters

Kalman filters combine traits of both HMMs and VARs: like VARs, they are linear Gaussian dBNs; but they also have latent variables like HMMs.

In their simplest form, Kalman Filters include a layer of one or more latent variables that model the unobservable part of the phenomenon,

$$\mathbf{Z}^{(t)} = A\mathbf{Z}^{(t-1)} + B\mathbf{U}^{(t)} + \zeta_t, \qquad \zeta_t \sim N(0, \Psi)$$

feeding into one or more observed variables

$$\mathbf{X}^{(t)} = C\mathbf{Z}^{(t)} + D\mathbf{U}^{(t)} + \varepsilon_t, \qquad \varepsilon_t \sim N(0, \Sigma)$$

with independent Gaussian noise added in both layers. Both layers often include additional (continuous) explanatory variables $\mathbf{U}$ and can also be augmented with (discrete) switching variables to allow for different regimes. If we exclude the latter, the assumption is that the system is jointly Gaussian: that makes it possible to frame Kalman filters as dBNs in the same way we did for VARs.

Relevant Functions in **bnlearn**

- creating BNs: `model2network()` `custom.fit()`.
- plotting DAGs: `graphviz.plot()`, along with `nodeRenderInfo()`, `edgeRenderInfo()` and `renderGraph()` from the **Rgraphviz** package.
- plotting BNs: `graphviz.chart()`.

Borrowed from **rstan**: `stan_model()` and `sampling()`.

Summary and Remarks

- BNs can take quite different forms depending on what assumptions we make on their local/global distributions and on their structure.
- Discrete BNs (DBNs) model categorical data using conditional probability tables (or equivalently multinomial logistic regressions).
- Gaussian BNs (GBNs) model continuous data as multivariate normal distributions (or equivalently linear regressions).
- Conditional Gaussian BNs (CGBNs) model mixed discrete and continuous data as a mixture model (like a mixed-effects model).
- General BNs are essentially hierarchical Bayesian models.
- Dynamic BNs (dBNs) model a variety of time series and stochastic processes in discete time.

Next:

- How do we make a computer system answer questions using a BN?

Thanks!

Any questions?