Towards the holy grail in machine learning

James Cussens, University of York

CP'18, Lille, 2018-08-29

Outline

What is the holy grail in machine learning?

- The user just states:
  1. what they know,
  2. what they want,
  3. and which data they have.
- This is also the goal of *probabilistic programming* (of which more later)
- Claim: CP can help progress towards this holy grail.

The Bayesian approach

$$P(\theta|D) \propto P(\theta)P(D|\theta)$$

The Bayesian approach

$$P(\theta|D) \propto P(\theta)P(D|\theta)$$

$$P(M|D) \propto P(M)P(D|M) = P(M)\int_{\theta} P(D|M,\theta)P(\theta)d\theta$$

The Bayesian approach

$$P(\theta|D) \propto P(\theta)P(D|\theta)$$

$$P(M|D) \propto P(M)P(D|M) = P(M)\int_\theta P(D|M,\theta)P(\theta)d\theta$$

- In the Bayesian approach to machine learning (aka ‘statistical inference’), ‘learning’ is reduced to probabilistic inference.
- Once a prior, likelihood (and perhaps a loss function) has been chosen, this is an entirely deductive process.

Optimisation problems in Bayesian statistics

$$P(M|D) \propto P(M)P(D|M)$$

- The MAP problem: $M^* = \arg\max_M P(M|D)$
- Decision problem: $A^* = \arg\min_A \sum_M L(A,M)P(M|D)$

Optimisation problems in Bayesian statistics

$$P(M|D) \propto P(M)P(D|M)$$

- The MAP problem: $M^* = \arg\max_M P(M|D)$
- Decision problem: $A^* = \arg\min_A \sum_M L(A, M)P(M|D)$

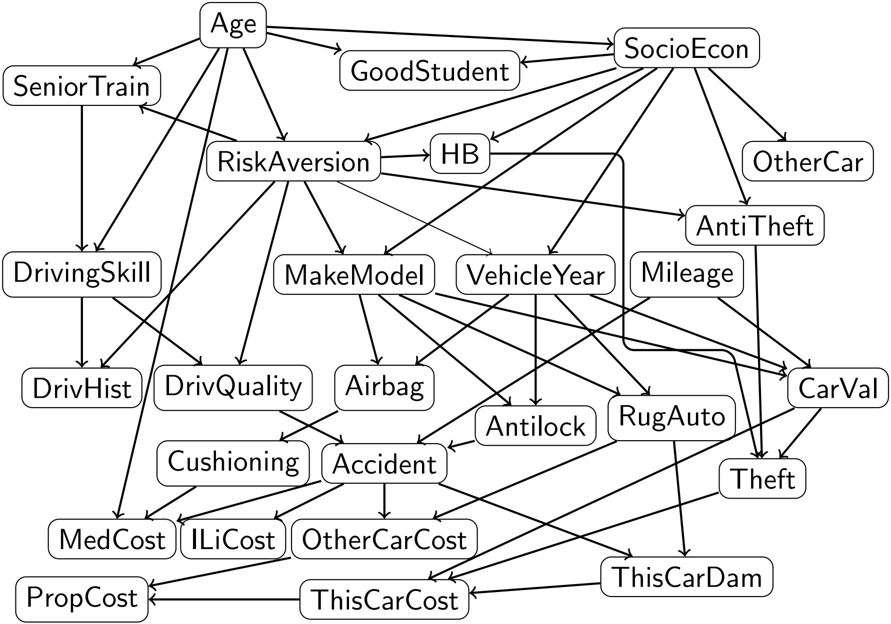
- **NB** The 'optimal' solution depends on our choice of prior.

The holy grail in (Bayesian) machine learning

- Encode all available knowledge/assumptions in the prior and likelihood (and loss function).
- Get some system to find a MAP model, *k*-best models, choose the best action, somehow represent the posterior, whatever.

The role of constraints

- Bayesian inference is easy when the prior is *conjugate*.
- So when estimating the probability of a coin landing heads we typically opt for a Beta distribution, just because it is easier.
- For priors over structures, such as graphs, a uniform distribution is the easy option.
- But in real applications we often have substantive domain knowledge.
- Constraints can be very helpful here.


Age
SocioEcon
GoodStudent
SeniorTrain
RiskAversion
HB
OtherCar
AntiTheft
DrivingSkill
MakeModel
VehicleYear
Mileage
DrivHist
DrivQuality
Airbag
CarVal
Antilock
RugAuto
Cushioning
Accident
Theft
MedCost
ILiCost
OtherCarCost
ThisCarDam
PropCost
ThisCarCost

Some discrete data

```
Age SocioEcon RiskAversion SeniorTrain VehicleYear ...
3 4 4 2 2 4 5 5 2 4 2 3 3 3 2 4 4 4 4 4 2 3 2 4 4 2 4
1 1 2 0 1 3 0 1 0 0 0 0 2 2 0 0 0 3 3 3 1 1 0 3 3 0 3
2 2 1 1 0 3 2 0 1 2 0 1 0 0 1 2 2 3 3 3 1 2 1 1 3 0 3
1 0 2 0 1 1 1 1 0 2 0 1 2 2 0 3 3 2 3 2 1 0 0 1 3 0 0
1 1 0 0 1 2 0 3 0 0 0 0 1 1 0 2 2 3 3 2 1 0 0 3 3 0 3
2 2 1 0 1 1 3 1 1 2 0 0 1 1 0 2 2 3 3 3 1 2 0 1 3 0 3
....
```

A Bayesian approach

$$P(M|D) \propto P(M) \int_{\theta} P(D|M,\theta)P(\theta)d\theta$$

- Choose (for the time being) a uniform prior for $P(M)$
- Choose the $P(\theta)$ so the integral (the *marginal likelihood*) has a closed form

A Bayesian approach

$$P(M|D) \propto P(M) \int_{\theta} P(D|M,\theta)P(\theta)d\theta$$

- Choose (for the time being) a uniform prior for $P(M)$
- Choose the $P(\theta)$ so the integral (the *marginal likelihood*) has a closed form
- From now on write $P(G)$ rather than $P(M)$ to emphasise that the model is a graph.

The BDeu score

Given complete discrete data D, with an appropriate choice of Dirichlet priors for the parameters, the log marginal likelihood for BN structure G with variables $i = 1, \ldots, p$ is:

$$\log P(D|G) = \sum_{i=1}^{p} c_i(G)$$

where $c_i(G)$, the *local score* for variable i depends only on the parents variable i has in graph G.

The BDeu score

Given complete discrete data D, with an appropriate choice of Dirichlet priors for the parameters, the log marginal likelihood for BN structure G with variables $i = 1, \ldots, p$ is:

$$\log P(D|G) = \sum_{i=1}^{p} c_i(G)$$

where $c_i(G)$, the *local score* for variable i depends only on the parents variable i has in graph G.

$$c_i(G) = c_{i \leftarrow \mathrm{Pa}_G(i)} = \sum_{j=1}^{q_i(G)} \left(\log \frac{\Gamma(\alpha_{ij})}{\Gamma(n_{ij} + \alpha_{ij})} + \sum_{k=1}^{r_i} \log \frac{\Gamma(n_{ijk} + \alpha_{ijk})}{\Gamma(\alpha_{ijk})} \right)$$

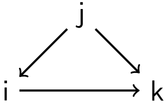
Combinatorial optimisation

With the preceding assumptions the BN model selection problem is to find a G^* such that:

$$G^* = \arg\max_G [\log P(D|G)] = \arg\max_G \left[\sum_{i=1}^{p} c_{i \leftarrow \mathrm{Pa}_G(i)} \right]$$

- This is a problem of *combinatorial optimisation*,
- which is known to be NP-hard.

Encoding graphs as real vectors

- The key to the integer programming (IP) (and CP) approach to BN model selection is to view digraphs as points in $\mathbb{R}^n$.
- We do this via *family variables*.
- This digraph: 

is this point in $\mathbb{R}^{12}$:

$i \leftarrow \{\}$	$i \leftarrow \{j\}$	$i \leftarrow \{k\}$	$i \leftarrow \{j,k\}$
0	1	0	0
$j \leftarrow \{\}$	$j \leftarrow \{i\}$	$j \leftarrow \{k\}$	$j \leftarrow \{i,k\}$
1	0	0	0
$k \leftarrow \{\}$	$k \leftarrow \{i\}$	$k \leftarrow \{j\}$	$k \leftarrow \{i,j\}$
0	0	0	1

BDeu scores as linear objective

Let $x(G)$ be the vector for digraph G, then

$$\log P(D|G) = \sum_{i=1}^{p} c_{i \leftarrow \mathrm{Pa}_G(i)} = \sum_{i=1}^{p} \sum_{J: i \notin J} c_{i \leftarrow J} x(G)_{i \leftarrow J}$$

The optimisation problem then becomes: find x^* such that

1. $x^* = \arg\max cx$
2. subject to x^* representing an acyclic directed graph.

BDeu scores as linear objective

Let $x(G)$ be the vector for digraph G, then

$$\log P(D|G) = \sum_{i=1}^{p} c_{i \leftarrow \mathrm{Pa}_G(i)} = \sum_{i=1}^{p} \sum_{J : i \notin J} c_{i \leftarrow J} x(G)_{i \leftarrow J}$$

The optimisation problem then becomes: find x^* such that

1. $x^* = \arg\max cx$
2. subject to x^* representing an acyclic directed graph.

- What's the problem here?

BDeu scores as linear objective

Let $x(G)$ be the vector for digraph G, then

$$\log P(D|G) = \sum_{i=1}^{p} c_{i \leftarrow \mathrm{Pa}_G(i)} = \sum_{i=1}^{p} \sum_{J : i \notin J} c_{i \leftarrow J} x(G)_{i \leftarrow J}$$

The optimisation problem then becomes: find x^* such that

1. $x^* = \arg\max cx$
2. subject to x^* representing an acyclic directed graph.

- What's the problem here?
- Too many $x(G)_{i \leftarrow J}$ variables!

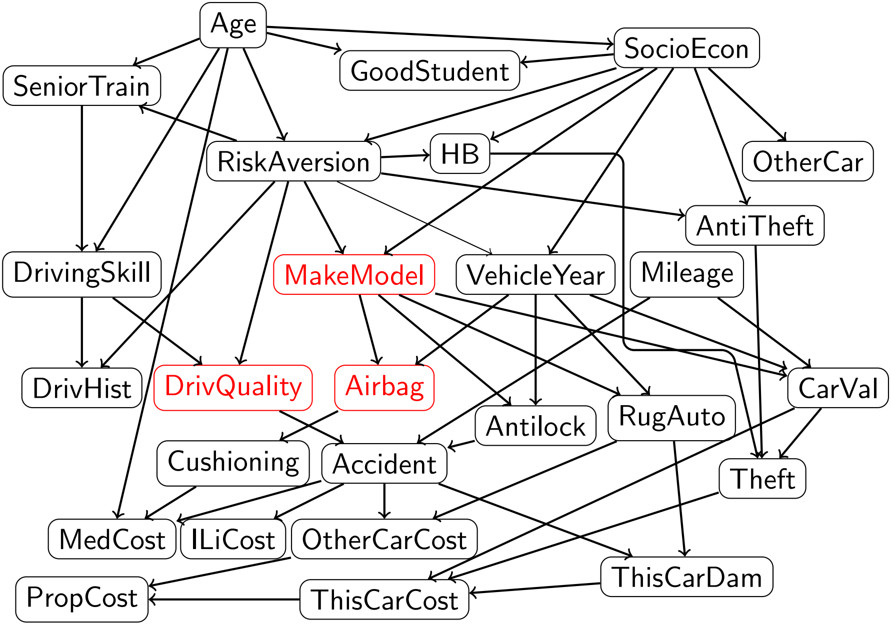
The integer program

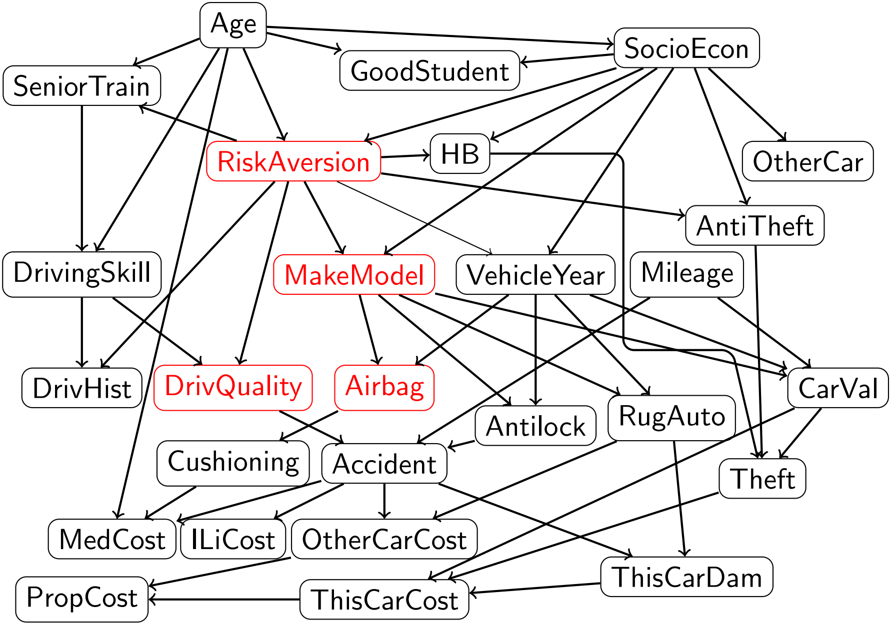
We can ensure that x represents an acyclic digraph with two classes of linear constraints and an integrality constraint.

1. 'convexity' $\forall i : \sum_J x_{i \leftarrow J} = 1$
2. 'cluster' $\forall C : \sum_{i \in C} \sum_{J \cap C = \emptyset} x_{i \leftarrow J} \geq 1$
3. x is a zero-one vector

We have an *integer program*: $\max cx$ subject to the above constraints. It is an IP since:

- the objective function is linear
- there are only linear and integrality constraints


Age
SocioEcon
GoodStudent
SeniorTrain
RiskAversion
HB
OtherCar
AntiTheft
DrivingSkill
MakeModel
VehicleYear
Mileage
DrivHist
DrivQuality
Airbag
CarVal
Antilock
RugAuto
Cushioning
Accident
Theft
MedCost
ILiCost
OtherCarCost
ThisCarDam
PropCost
ThisCarCost

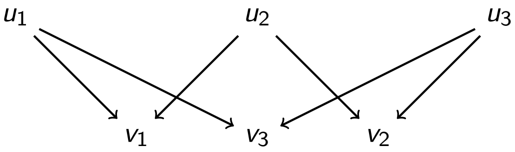

Age
SocioEcon
GoodStudent
SeniorTrain
RiskAversion
HB
OtherCar
AntiTheft
DrivingSkill
MakeModel
VehicleYear
Mileage
DrivHist
DrivQuality
Airbag
CarVal
Antilock
RugAuto
Cushioning
Accident
Theft
MedCost
ILiCost
OtherCarCost
ThisCarDam
PropCost
ThisCarCost

Why IP/CP for BNSL?

- There are very many 'search and score' algorithms for BNSL.
- Hillclimbing is a common choice
- So what are the pros (and cons) of using IP/CP? [Cus11, vBH15]
- A big win is that we can add constraints without needing to come up with a new algorithm.

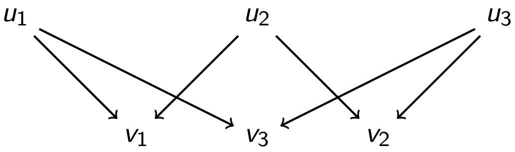
Necessary constraints in pedigrees

This subgraph can never occur in a DAG representing a *pedigree* ('family tree')



Necessary constraints in pedigrees

This subgraph can never occur in a DAG representing a *pedigree* ('family tree')



So let, $I_f(u)$ indicate that u is a female, then:
At most one mother:

$$\forall u, v, w : I(\{u, w\} \to v) + I_f(u) + I_f(w) \leq 2$$

At least one mother:

$$\forall u, v, w : I(\{u, w\} \to v) - I_f(u) - I_f(w) \leq 0$$

Digression: Exploiting the solver

- I'm advocating a *declarative approach* to machine learning: declare what we know and leave the rest to the solver.
- A nice plus: many IP solvers (CPLEX, Gurobi and SCIP) allow you to find:
  - the k-best solutions
  - all solutions with objective value above some threshold
- If getting an optimal solution is impractical, you at least get an optimality gap.
- Moreover, CPLEX and Gurobi (and SCIP using UG) will grab available cores with zero effort from the user.

Other constraints for BNSL

1. There must (not) be an arrow from A to B
2. There must (not) be a path from A to B
3. If A and B are co-parents there must be an edge between them (chordality).
4. The graph must satisfy certain conditional independence relations.

- All easy to throw in.

Other constraints for BNSL

1. There must (not) be an arrow from A to B
2. There must (not) be a path from A to B
3. If A and B are co-parents there must be an edge between them (chordality).
4. The graph must satisfy certain conditional independence relations.

- All easy to throw in.
- But not always easy to ensure fast solving!

Why 'cluster' constraints?

$$\forall C : \sum_{i \in C} \sum_{J \cap C = \emptyset} x_{i \leftarrow J} \geq 1$$

- One can rule out cycles in graphs with a quadratic number of linear constraints (and a linear number of additional variables).
- We choose to use exponentially many cluster constraints [JSGM10] since they are *facets* of the convex hull of directed acyclic graphs, leading to a tight linear relaxation.
- They are added as *cutting planes* 'on the fly'. (Finding one is NP-hard; a sub-IP is used.)
- Studený [Stu15] showed that there is a facet associated with every *connected matroid*. Cluster constraints are a special case.

Not all facets are equal

- We computed all the 135 facets of the convex hull of the 543 DAGs when there are only 4 BN variables. Ones like this improved performance far more than others. Why?

$$
\begin{aligned}
& x_{a\leftarrow\{b,c\}} + x_{a\leftarrow\{b,d\}} + x_{a\leftarrow\{b,c,d\}} + \\
& x_{b\leftarrow\{a,c\}} + x_{b\leftarrow\{a,d\}} + x_{b\leftarrow\{a,c,d\}} + \\
& x_{c\leftarrow\{d\}} + x_{c\leftarrow\{a,b\}} + x_{c\leftarrow\{a,d\}} + x_{c\leftarrow\{b,d\}} + x_{c\leftarrow\{a,b,d\}} + \\
& x_{d\leftarrow\{c\}} + x_{d\leftarrow\{a,b\}} + x_{d\leftarrow\{a,c\}} + x_{d\leftarrow\{b,c\}} + x_{d\leftarrow\{a,b,c\}} \leq 2
\end{aligned}
$$

Not all facets are equal

- We computed all the 135 facets of the convex hull of the 543 DAGs when there are only 4 BN variables. Ones like this improved performance far more than others. Why?

$$
\begin{aligned}
&x_{a\leftarrow\{b,c\}} + x_{a\leftarrow\{b,d\}} + x_{a\leftarrow\{b,c,d\}} + \\
&x_{b\leftarrow\{a,c\}} + x_{b\leftarrow\{a,d\}} + x_{b\leftarrow\{a,c,d\}} + \\
&x_{c\leftarrow\{d\}} + x_{c\leftarrow\{a,b\}} + x_{c\leftarrow\{a,d\}} + x_{c\leftarrow\{b,d\}} + x_{c\leftarrow\{a,b,d\}} + \\
&x_{d\leftarrow\{c\}} + x_{d\leftarrow\{a,b\}} + x_{d\leftarrow\{a,c\}} + x_{d\leftarrow\{b,c\}} + x_{d\leftarrow\{a,b,c\}} \leq 2
\end{aligned}
$$

- This facet is *score-equivalent*. If two BNs are Markov equivalent then the LHS of the facet is the same for both BNs.
- And we typically use objectives that are score-equivalent.
- The facet above corresponds to the (connected) matroid whose *circuits* are $\{\{a,b,c\},\{a,b,d\},\{c,d\}\}$.

The too-many variables problem

- Problem: We can only fit so many $x_{i \leftarrow J}$ family variables into the solver.
- 'Pruning' is used to delete many, but we can still end up with too many.

The too-many variables problem

- Problem: We can only fit so many $x_{i \leftarrow J}$ family variables into the solver.
- 'Pruning' is used to delete many, but we can still end up with too many.
- Very few $x_{i \leftarrow J}$ family variables have non-zero value in any solution.

The too-many variables problem

- Problem: We can only fit so many $x_{i \leftarrow J}$ family variables into the solver.
- 'Pruning' is used to delete many, but we can still end up with too many.
- Very few $x_{i \leftarrow J}$ family variables have non-zero value in any solution.
- Solution: Create IP variables on the fly using a *pricer*.
- This is the dual of adding cutting planes ('constraints on the fly').
- The implementation requires a lot of bookkeeping :-(

An alternative approach to BNSL

- I have just presented a *search and score* approach to BNSL.
- The other main approach is known as *constraint-based* BN learning.
- (There are also hybrid approaches.)

Constraint-based BN learning

- A Bayesian network encodes a set of *conditional independence relations*.
- Is A independent of B given C? ($A \perp B|C$?)
- So ask which conditional independence relations hold and which do not and then view the answers as *constraints* on an acceptable DAG.
- It may be that only a DAG with *latent* (i.e. unobserved) and/or *selection* (i.e. conditioned on) variables satisfies all the constraints.
- Either use statistical tests on the data or pretend we have an oracle to answer these questions.
- For efficiency only do some tests.

CP for Constraint-based BN learning

- Constraint-based BN learning is a *constraint satisfaction* problem, . . .
- . . . albeit one where not all constraints are known at the outset.
- However the best-known algorithms for constraint-based BN learning (PC, FCI, RFCI) do not use CP methods.
- But CP based methods do exist: *Constraint-based Causal Discovery: Conflict Resolution with Answer Set Programming* [HEJ14]

Probabilistic Programming

From the Stan website:

"Users specify log density functions in Stan's probabilistic programming language and get:

- full Bayesian statistical inference with MCMC sampling (NUTS, HMC)
- approximate Bayesian inference with variational inference (ADVI)
- penalized maximum likelihood estimation with optimization (L-BFGS)"

The (log) density function is typically a posterior distribution.

Is Stan declarative or imperative?

"A Stan program defines a statistical model through a conditional probability function $p(\theta|y, x)$, where θ is a sequence of modeled unknown values (e.g., model parameters, latent variables, missing data, future predictions),
Stan is an imperative language, like C or Fortran (and parts of C++, R, Python, and Java), in the sense that is based on assignment, loops, conditionals, local variables, object-level function application, and array-like data structures."

Constrained optimisation in Machine Learning

- Constrained optimisation is everywhere in Machine Learning.
- For example, using Support Vector Machines involves solving a quadratic programming problem (which is typically not solved by sending the problem to a QP solver).
- Sometimes CP methods explicitly used, particularly for: clustering, frequent item-set mining, BNSL/causal inference.
- See the AIJ Special Issue [PTG17] for example.

MiningZinc: A Language for Constraint-based Mining

- "MiningZinc is a high-level language for constraint-based mining that supports both user-defined constraints and efficient, specialised solving. It consists of a language and a framework."
- Can be applied to e.g. frequent itemset mining and clustering.
- "The language is standard MiniZinc."
- "It supports both generic CP, SAT and MIP solvers, as well as specialised constraint-based mining systems."

Constraints and the democratisation of machine learning

- If we can simply declare what the learning task is, can the process of machine learning be automated (and de-skilled)?
- Consider the SYNTH project: "What we want to do in the SYNTH project is to automate a subfield of AI itself. That field is data science, ..."
- Will the end-user understand what has happened?
- Need to distinguish how to optimise (computational) from what to optimise (statistical).

James Cussens.
Bayesian network learning with cutting planes.
In Fabio G. Cozman and Avi Pfeffer, editors, *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI 2011)*, pages 153–160, Barcelona, 2011. AUAI Press.

A. Hyttinen, F. Eberhardt, and M. Järvisalo.
Constraint-based causal discovery: Conflict resolution with answer set programming.
In *Proc. UAI 2014*, 2014.

Tommi Jaakkola, David Sontag, Amir Globerson, and Marina Meila.
Learning Bayesian network structure using LP relaxations.
In *Proceedings of 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 358–365, 2010.
Journal of Machine Learning Research Workshop and Conference Proceedings.

Andrea Passerini, Guido Tack, and Tias Guns.
Special issue on combining constraint solving with mining and learning.
Artificial Intelligence, 244:1–396, March 2017.

Milan Studený.
How matroids occur in the context of learning Bayesian network structure.
In Marina Meila and Tom Heskes, editors, *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence (UAI 2015)*, pages 832–841. AUAI Press, 2015.

Peter van Beek and Hella-Franziska Hoffmann.
Machine learning of Bayesian networks using constraint programming.
In *Proc. CP 2015*, August 2015.