Optimal Algorithms for Learning Bayesian Network Structures
Integer Linear Programming and Evaluations

James Cussens, University of York

UAI, 2015-07-12

Encoding digraphs as real vectors

- The key to the integer programming (IP) approach to BN model selection is to view digraphs as points in $\mathbb{R}^n$.
- We do this via *family variables*.
- This digraph: $$\begin{tikzcd} & j \arrow[dl] \arrow[dr] & \\ i \arrow[rr] & & k \end{tikzcd}$$ 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

Encoding digraphs as real vectors

- The key to the integer programming (IP) approach to BN model selection is to view digraphs as points in $\mathbb{R}^n$.
- We do this via *family variables*.
- This digraph: $$\begin{tikzcd} & \mathrm{j} \arrow[dl] \arrow[dr] & \\ \mathrm{i} \arrow[rr] & & \mathrm{k} \end{tikzcd}$$ 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

A linear objective

Let $x(G)$ be the vector for digraph G, then for a decomposable score:

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

The ('vanilla') optimisation problem then becomes: find $\check{x}$ such that

1. $\check{x} = \arg\max cx$
2. and $\check{x}$ represents an acyclic digraph.

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

Relaxation

Solving the following *relaxation* of the problem is very easy

1. $\forall i : \sum_J x_{i \leftarrow J} = 1$
2. $\cancel{\forall C : \sum_{i \in C} \sum_{J \cap C = \emptyset} x_{i \leftarrow J} \geq 1}$ (combinatorial relaxation)
3. ~~x is a zero-one vector~~ (linear relaxation)

Relaxations:

- provide an upper bound on an optimal solution,
- and we might 'get lucky' and find that the solution to the relaxation satisfies all the constraints of the original problem.

Tightening the relaxation

- We tighten the relaxation by adding *cutting planes*
- Let x^* be the solution to the current relaxation,
- If $\sum_{i \in C} \sum_{J \cap C = \emptyset} x^*_{i \leftarrow J} < 1$ then the valid inequality $\sum_{i \in C} \sum_{J \cap C = \emptyset} x_{i \leftarrow J} \geq 1$ is added to get a new relaxation,
- and so on.

- This procedure improves the upper bound (the ‘dual bound’).
- We might get lucky and find that x^* represents an acyclic digraph, in which case the problem is solved.
- We use the SCIP system which will find additional non-problem-specific cutting planes as well.

The separation problem

The *separation problem* is:

- Given x^* (the solution to the current LP relaxation),
- Find C such that $\sum_{i \in C} \sum_{J \cap C = \emptyset} x^*_{i \leftarrow J} < 1$, or show that no such C exists.

- This separation problem has recently been shown to be NP-hard [CJKB15].
- In the GOBNILP system a sub-IP is used to solve it.
- Note: the vast majority of cluster inequalities are **not** added, since they do not tighten the relaxation.

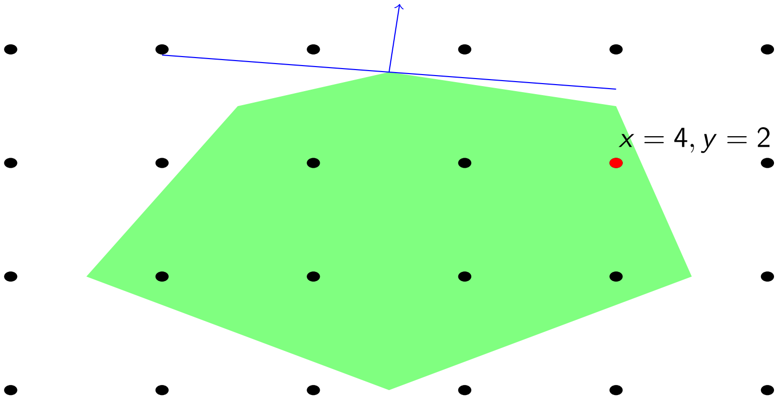
Getting lucky . . . eventually

Eskimo pedigree. 1614 BN variables. At most 2 parents. Simulated genotypes. 11934 IP variables. Old version of GOBNILP.

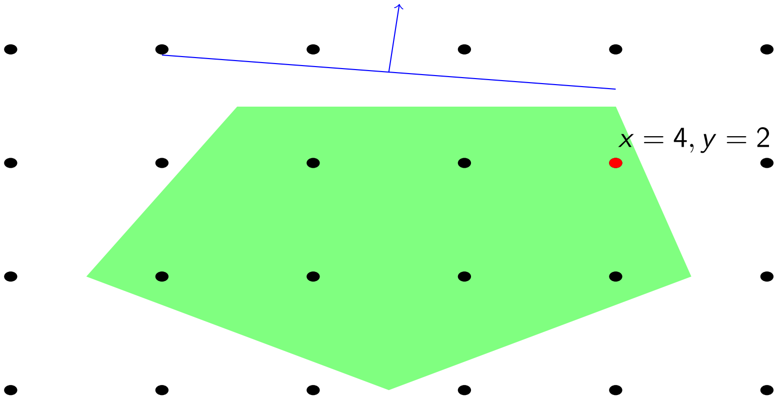
```
 time |frac|cuts |   dualbound    | primalbound   |  gap
 1110s|120 | 661 | -3.162149e+04 |-4.616035e+04 |  45.98%
 1139s|118 | 669 | -3.162175e+04 |-4.616035e+04 |  45.98%
 1171s| 94 | 678 | -3.162213e+04 |-4.616035e+04 |  45.97%
 1209s| 26 | 684 | -3.162220e+04 |-4.616035e+04 |  45.97%
 1228s|103 | 685 | -3.162223e+04 |-4.616035e+04 |  45.97%
 1264s|  0 | 692 | -3.162234e+04 |-4.616035e+04 |  45.97%
*1266s|  0 |   - | -3.162234e+04 |-3.162234e+04 |   0.00%

SCIP Status          : problem is solved [optimal solution founc
Solving Time (sec) : 1266.40
```

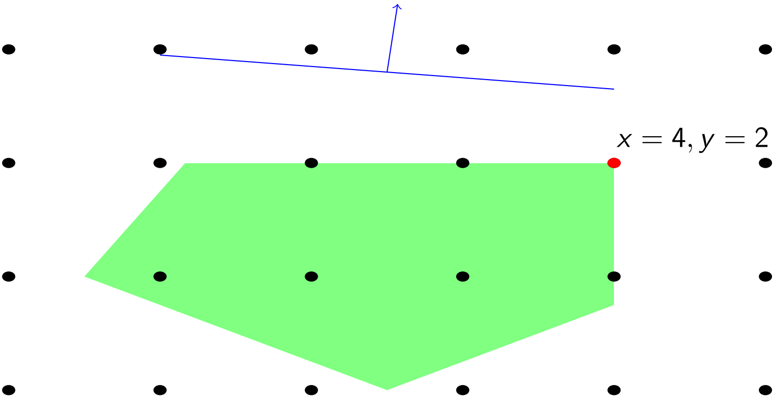
Cutting planes in two dimensions



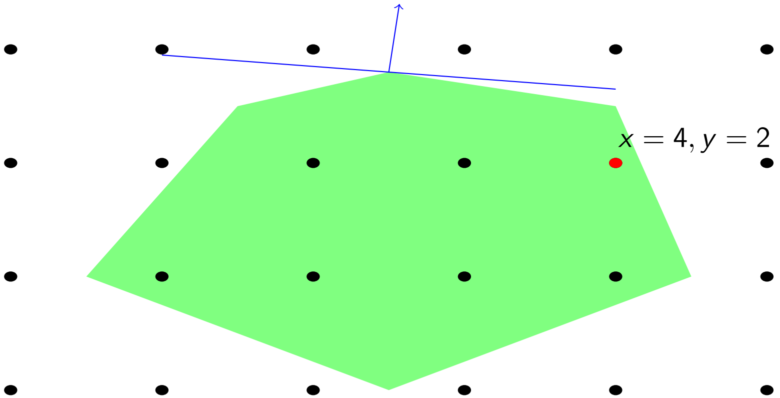
Cutting planes in two dimensions



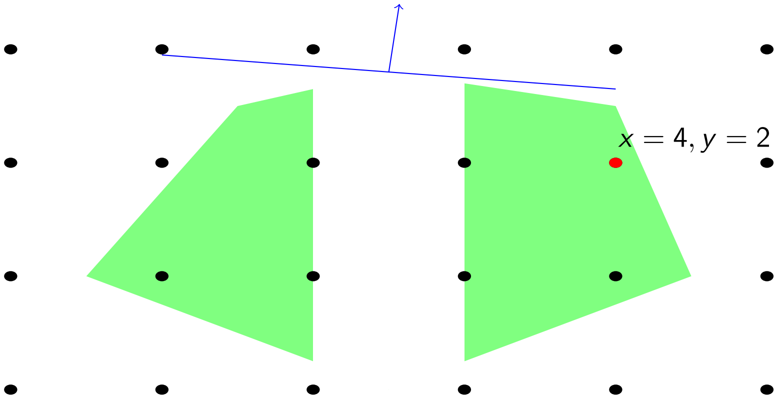
Cutting planes in two dimensions



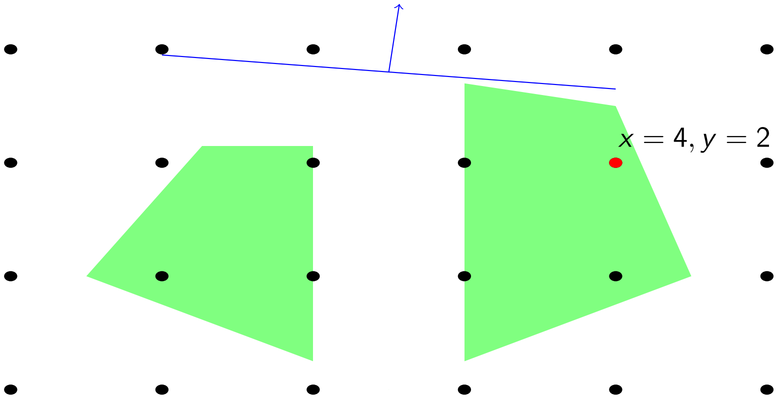
Branch-and-cut



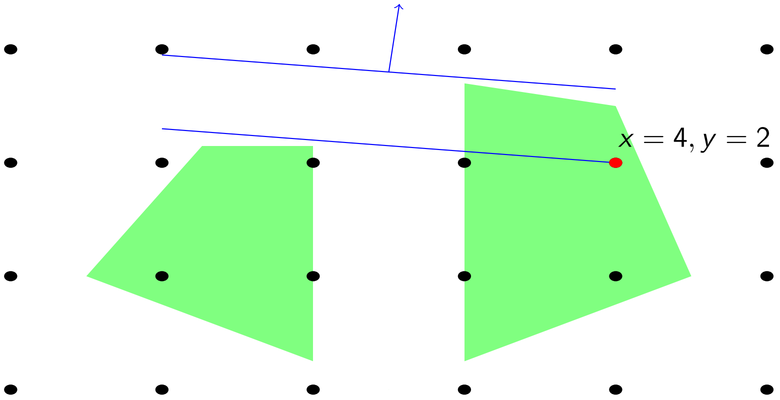
Branch-and-cut



Branch-and-cut



Branch-and-cut



Branch and cut

For any node in the search tree (including the root) ...

```
1. Let x* be the LP solution.
2. If x* worse than incumbent then exit.
3. If there are valid linear inequalities
        not satisfied by x*
        add them and go to 1.
   Else if x* is integer-valued then
        the node is solved
   Else branch on a variable with
        non-integer value in x*
        to create two child nodes
        (propagate if possible)
```

The convex hull

- Since each acyclic digraph is a point in $\mathbb{R}^n$ there is a convex hull of acyclic digraphs.
- If our IP had all the inequalities defining this convex hull we could drop the integrality restriction and solve the problem with a *linear program* (LP).
- An LP, unlike, an IP, can be solved in polynomial time.
- For 4 BN variables, there are 543 acyclic digraphs (living in $\mathbb{R}^{28}$) and the convex hull is defined by 135 inequalities.

Facets

- The inequalities defining the convex hull are called *facets*.
- We have shown [CJKB15, CHS15] that the cluster inequalities, first introduced by [JSGM10], are facets.
- But there are very many other facets, for example this one for BN variable set $\{a, b, c, d\}$:

$$
\begin{aligned}
& x_{a\leftarrow bc} + x_{a\leftarrow bd} + x_{a\leftarrow cd} + 2x_{a\leftarrow bcd} \\
+ & x_{b\leftarrow ac} + x_{b\leftarrow ad} + x_{b\leftarrow acd} \\
+ & x_{c\leftarrow ab} + x_{c\leftarrow ad} + x_{c\leftarrow abd} \\
+ & x_{d\leftarrow ab} + x_{d\leftarrow ac} + x_{d\leftarrow abc} \qquad \leq 2
\end{aligned}
$$

Characteristic imsets and matroids

- An alternative approach—*characteristic imsets*, developed by Milan Studený—encodes each Markov equivalence class of BNs as a zero-one vector [CHS15].

$$\mathbf{c}(S) = \sum_{i \in S} \sum_{S \setminus \{i\} \subseteq J} x_{i \leftarrow J}$$

- At this conference Studený has a paper which uses matroid theory to derive useful results for both the c-imset and family-variable polytope [Stu15].
- Milan's paper generalises the proof that 'cluster' inequalities are facets.

Strong branching

- Which variable to branch on?

Strong branching

- Which variable to branch on?
- SCIP's default approach aims (mainly) to improve the 'dual bound' on both sides of the branch.
- *Strong branching* tries out candidate variables before choosing which one to branch on.
- This is expensive (lots of LP solving) so done mainly at the top of the search tree.

Propagation

- Alternatively, one can aim for lots of propagation.
- If $x_{i \leftarrow \{j,k\}} = 1$ and $x_{k \leftarrow \{\ell\}} = 1$ then we can set e.g. $x_{\ell \leftarrow \{i\}}$ to 0.
- van Beek and Hoffmann [vBH15] have recently applied a constraint programming approach to BN learning which uses auxiliary variables and lots of propagation.

GOBNILP approach

In the latest version of GOBNILP ...

- We start branching if adding cutting planes has made little progress for 10 rounds (`separating/maxstallrounds = 10`)
- We have auxiliary variables representing both directed and undirected edges of the DAG.
- We branch on these variables (not the family variables).
- We use SCIP's default branching rule (`relpscost`) with some non-default parameter values.

Constraint integer programming (SCIP)

- Branch-and-cut is a ‘declarative’ algorithm.
- It treats e.g. the acyclicity constraint handler as (almost) a black box.
- So we can add in additional constraints, if we have them, without having to come up with a new algorithm.

Conditional independence constraints

- Recall the acyclicity constraint (cluster inequality): $\forall C : \sum_{i \in C} \sum_{J \cap C = \emptyset} x_{i \leftarrow J} \geq 1$
- Suppose for some C' we have $\sum_{i \in C'} \sum_{J \cap C' = \emptyset} x_{i \leftarrow J} = 1$
- Then the BN nodes in C' have a common ancestor in C' and are thus *d-connected*.
- So suppose we want $j \perp k$, then $\forall C : \{j, k\} \subseteq C \Rightarrow \sum_{i \in C} \sum_{J \cap C = \emptyset} x_{i \leftarrow J} \geq 2$
- GOBNILP's conditional independence constraint handler provides such inequalities as cutting planes.

Other constraints

- We can add constraints to rule out *immoralities* to learn decomposable models, but Kangas *et al* [KNK14] do better!
- Oates *et al* [OSMC15] learned multiple BNs (from multiple datasets) with a penalty for structural differences.

Too many variables!

- GOBNILP generates all its IP variables before it starts the solving process.
- With too many it will just crash, and it gets progressively slower with more IP variables.
- It is not the parent set size limit *per se* which is the limiting factor, since, by creating fake BN nodes, one can encode **any** BN learning problem as one with a limit of at most two parents: replace $x_{i \leftarrow j,k,\ell}$ with $x_{i \leftarrow j\&k,\ell}$, set $x_{j\&k \leftarrow \{j\}} = x_{j\&k \leftarrow \{k\}} = 1$.

Column generation

- Column generation = variable generation
- In the column generation approach new variables are created only if setting them to a non-zero value **raises** the upper ('dual') bound.
- This is the dual to adding cutting planes which **lower** the upper bound.
- The resulting algorithm is *branch-price-and-cut*.

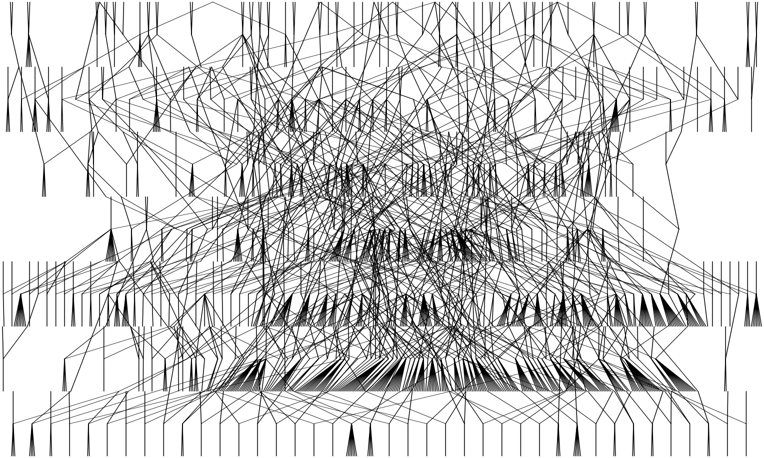
Empirical evaluations

- Now for some empirical evaluations . . .

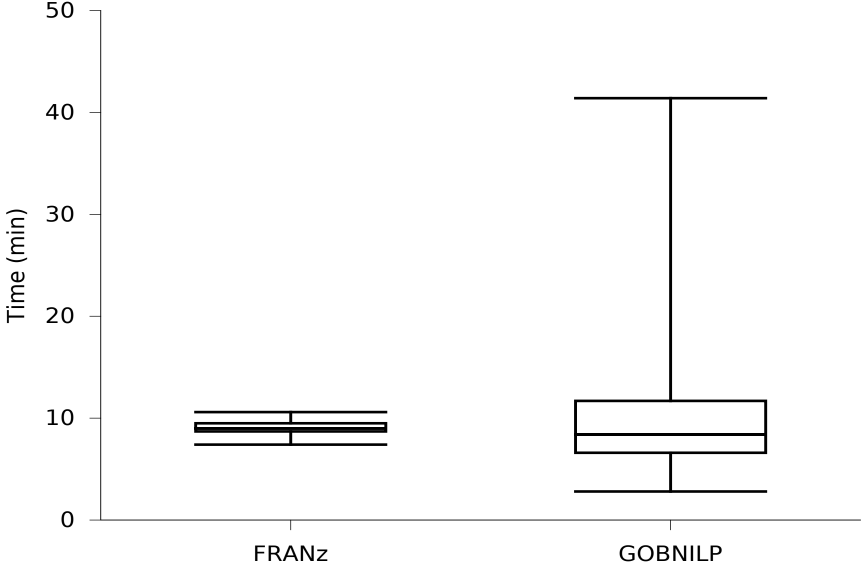
Pedigree learning with GOBNILP

- GOBNILP's main (funded!) target problem has been *pedigree learning*.
- In a pedigree there are at most two parents: a known father and a known mother.
- So even with very many individuals in the pedigree ($=$ BN nodes) there are not so many IP variables.

1614 node 'Polar Eskimo Genealogy'



FRANz vs GOBNILP: Eskimo pedigree solving times



FRANz vs GOBNILP: Eskimo pedigree accuracy

	GOBNILP	FRANz
Precision	95.2%	94.1%
Recall	96.8%	95.4%

- See Sheehan *et al* [SBC14] for further details.

GOBNILP for general BN learning

- Plenty of empirical results on the GOBNILP webpage `https://www.cs.york.ac.uk/aig/sw/gobnilp/`.
- Those results all ask SCIP to use CPLEX to solve the linear relaxations—that makes a difference!

GOBNILP with no parent set restriction

Name	p	n	IPVars	ScoreTime	SolveTime/Gap
Adult	14	30162	3546	4	11.2
Wine	14	178	790	1	2.8
Letter	17	20000	83961	100	0.88%
Zoo	17	101	3590	3	97.4
Voting	17	435	801	18	1.7
Statlog	19	752	4899	56	28.0
Hepatitis	20	126	972	64	2.3
Image	20	2310	13713	249	332.6
Imports	23	205	13396	694	287.2
Meta	23	527	FAIL	FAIL	FAIL
Mushroom.1000	23	1000	25697	1124	5.65%
Mushroom	23	8124	FAIL	FAIL	FAIL
Heart	23	212	631	1274	0.6
Horse.23	23	300	925	1910	2.0
Parkinsons	23	195	3699	1166	4.8

GOBNILP with no parent set restriction

- Datasets on the preceding slide downloaded from `urlearning.org` and mostly originate from UCI.
- GOBNILP failed during scoring on all the following larger datasets: Sensors, Autos, Horse, SteelPlates, Alarm.1000, Flag, Epigenetics, Wdbc, Soybean, Water, Bands, Spectf and LungCancer.

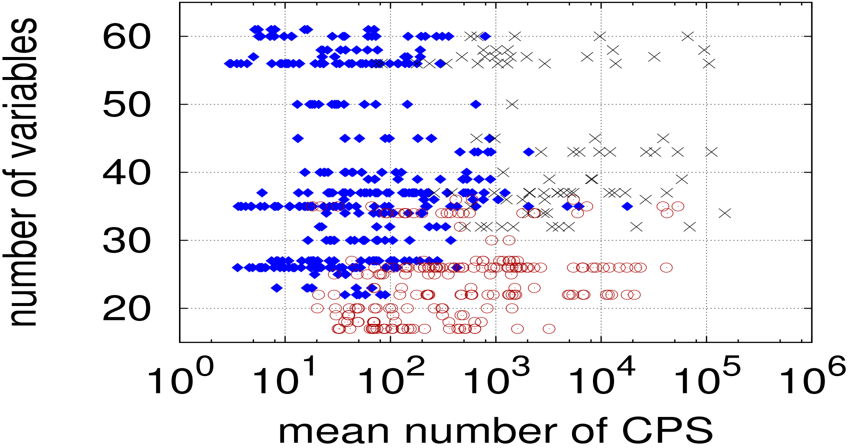
A CP approach to exact BN learning

- van Beek and Hoffmann [vBH15] have compared their algorithm *CPBayes* to GOBNILP 1.4.1 and A*.
- GOBNILP 1.6.1 does better that 1.4.1 (see GOBNILP page) but the trend is the same.

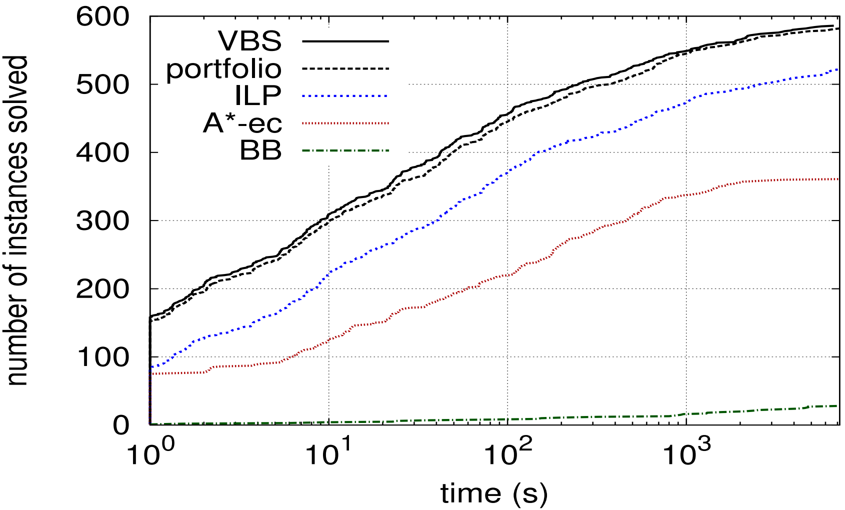
			BDeu				BIC			
				GOBN.	A*	CPBayes		GOBN.	A*	CPBayes
Benchmark	n	N	d	v1.4.1	v2015	v1.0	d	v1.4.1	v2015	v1.0
shuttle	10	58,000	812	58.5	0.0	0.0	264	2.8	0.1	0.0
adult	15	32,561	768	1.4	0.1	0.0	547	0.7	0.1	0.0
letter	17	20,000	18,841	5,060.8	1.3	1.4	4,443	72.5	0.6	0.2
voting	17	435	1,940	16.8	0.3	0.1	1,848	11.6	0.4	0.1
zoo	17	101	2,855	177.7	0.5	0.2	554	0.9	0.4	0.1
tumour	18	339	274	1.5	0.9	0.2	219	0.4	0.9	0.2
lympho	19	148	345	1.7	2.1	0.5	143	0.5	1.0	0.2
vehicle	19	846	3,121	90.4	2.4	0.7	763	4.4	2.1	0.5
hepatitis	20	155	501	2.1	4.9	1.1	266	1.7	4.8	1.0
segment	20	2,310	6,491	2,486.5	3.3	1.3	1,053	13.2	2.4	0.5
mushroom	23	8,124	438,185	OT	255.5	561.8	13,025	82,736.2	34.4	7.7
autos	26	159	25,238	OT	918.3	464.2	2,391	108.0	316.3	50.8
insurance	27	1,000	792	2.8	583.9	107.0	506	2.1	824.3	103.7
horse colic	28	300	490	2.7	15.0	3.4	490	3.2	6.8	1.2
steel	28	1,941	113,118	OT	902.9	21,547.0	93,026	OT	550.8	4,447.6
flag	29	194	1,324	28.0	49.4	39.9	741	7.7	12.1	2.6
wdbc	31	569	13,473	2,055.6	OM	11,031.6	14,613	1,773.7	1,330.8	1,460.5
water	32	1,000					159	0.3	1.6	0.6
mildew	35	1,000	166	0.3	7.6	1.5	126	0.2	3.6	0.6
soybean	36	266					5,926	789.5	1,114.1	147.8
alarm	37	1,000					672	1.8	43.2	8.4
bands	39	277					892	15.2	4.5	2.0
spectf	45	267					610	8.4	401.7	11.2
sponge	45	76					618	4.1	793.5	13.2
barley	48	1,000					244	0.4	1.5	3.4
hailfinder	56	100					167	0.1	9.9	1.5
hailfinder	56	500					418	0.5	OM	9.3
lung cancer	57	32					292	2.0	OM	10.5
carpo	60	100					423	1.6	OM	253.6
carpo	60	500					847	6.9	OM	OT

Which algorithm?

Which is faster, GOBNILP (blue) or A* (red), on a given instance [MKMJM14]?



Portfolio approach [MKMJM14]



Is optimal learning worth the effort?

Here are the main findings from Malone *et al* [MJM15] (to be presented at this conference)

- Bigger datasets result in BNs with better predictive liklelihood.
- "[Optimal learning] guarantees consistently translate into networks with good generalization. Algorithms with weaker guarantees produce networks with inconsistent generalization."

Acknowledgements

- GOBNILP has been supported by the UK Medical Research Council under grant G1002312.
- This tutorial supported by the UK National Centre for the Replacement, Refinement & Reduction of Animals in Research under grant NC/K001264/1.
- Thanks to Peter van Beek for discussions on CPBayes.

James Cussens, David Haws, and Milan Studený.
Polyhedral aspects of score equivalence in Bayesian network structure learning.
Arkiv 1503.00829, March 2015.

James Cussens, Matti Järvisalo, Janne H. Korhonen, and Mark Bartlett.
Polyhedral theory for Bayesian network structure learning.
in preparation, June 2015.

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.

Kustaa Kangas, Teppo Niinimäki, and Mikko Koivisto.

Learning chordal Markov networks by dynamic programming.
In *Proc. NIPS 2014*, 2014.

Brandon Malone, Matti Järvisalo, and Petri Myllymäki.
Impact of learning strategies on the quality of Bayesian networks: An empirical evaluation.
In *Proc. UAI 2015*, 2015.

Brandon Malone, Kustaa Kangas, Mikko Koivisto Matti Järvisalo, and Petri Myllymäki.
Predicting the hardness of learning Bayesian networks.
In Brodley and Stone, editors, *Proc. AAAI 2014*, 2014.

Chris Oates, Jim Smith, Sach Mukherjee, and James Cussens.
Exact estimation of multiple directed acyclic graphs.
Statistics and Computing, 2015.
Forthcoming.

Nuala Sheehan, Mark Bartlett, and James Cussens.

Improved maximum likelihood reconstruction of complex multi-generational pedigrees.
Theoretical Population Biology, 97:11–19, 2014.

Milan Studený.
How matroids occur in the context of learning Bayesian network structure.
In *Proc. UAI 2015*, 2015.

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