Recent work on improving Bayesian network structure learning using integer programming

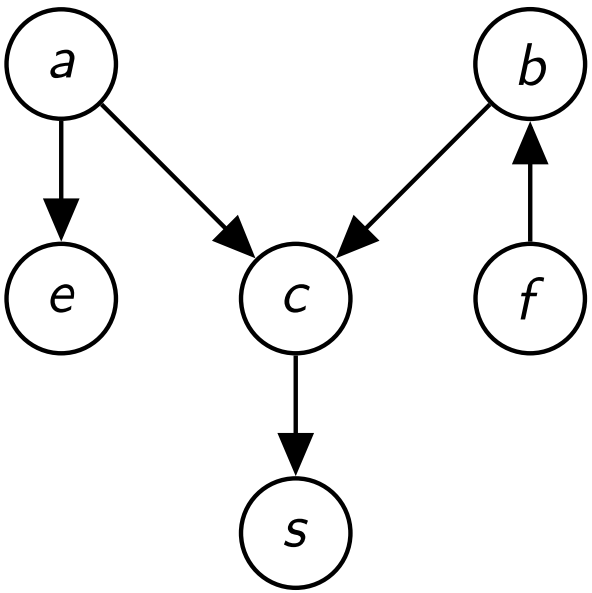
James Cussens

UTIA, 2026-31-08

Outline

Bayesian network structure learning

- Goal: From data drawn from some (unknown) multivariate probability distribution, find a Bayesian network which encodes the conditional independence relations which hold in that distribution.
- For example, if each datapoint were a vector of values for a, b, c, s, e, f we might 'learn' the following BN:



Constraint-based vs score-based BNSL

Constraint-based Estimate which CI relations hold in the unknown distribution using statistical tests and construct a BN that encodes these CI relations.

Score-based View the problem as statistical model selection: assign each BN a score (e.g. BIC or something Bayesian) and attempt to find the highest scoring BN.

These approaches are not fundamentally different. See, for example:
Conditions Under Which Conditional Independence and Scoring Methods Lead to Identical Selection of Bayesian Network Models, Cowell, UAI 2001.

Score-based BNSL as discrete optimisation

- Score-based BNSL is a particular sort of *discrete optimisation* ...
- ... find the best thing (e.g. graph, TSP tour, crew schedule) from a (typically large) finite set of things.
- So we can use existing methods and software for *constrained optimisation*.
- And exploit their flexibility to add constraints (other than the basic constraint of acyclicity).

Don't learn what you know

- Bayesian networks encode conditional independence (CI) constraints.
- In real-world BN learning, one typically knows a fair bit about the variables.
- Such knowledge should be added using constraints (rather than just hoping the data encapsulates it).
- Here we look at adding CI constraints to BN learning.

Checking CI constraints with propositional logic

At the very least we must be able to check whether a particular DAG satisfies a particular CI constraint.
If $A \perp B | S$ then A must be separated from B by S in $(\mathcal{G}_{\mathrm{An}(A \cup B \cup S)})^m$. So define these propositional symbols:

α_i	i is in $\mathrm{An}(A \cup B \cup S)$
$y_{i \leftarrow j}$	There is an arrow from j to i in $\mathcal{G}_{\mathrm{An}(A \cup B \cup S)}$
z_{i-j}	i and j are connected by a path in $(\mathcal{G}_{\mathrm{An}(A \cup B \cup S)})^m$ that avoids S.
λ	A and B are **not** separated by S in $(\mathcal{G}_{\mathrm{An}(A \cup B \cup S)})^m$

Rules for checking a CI constraint

$$x_{i \leftarrow j} \qquad i \leftarrow j \in \mathcal{G} \tag{1}$$

$$\alpha_i \qquad i \in (A \cup B \cup S) \tag{2}$$

$$\alpha_i \wedge x_{i \leftarrow j} \rightarrow \alpha_j \tag{3}$$

$$\alpha_i \wedge x_{i \leftarrow j} \rightarrow y_{i \leftarrow j} \tag{4}$$

$$\alpha_i \wedge x_{i \leftarrow j} \rightarrow z_{i-j} \qquad \{i, j\} \cap S = \emptyset \tag{5}$$

$$y_{i \leftarrow j} \wedge y_{i \leftarrow k} \rightarrow z_{j-k} \qquad \{j, k\} \cap S = \emptyset \tag{6}$$

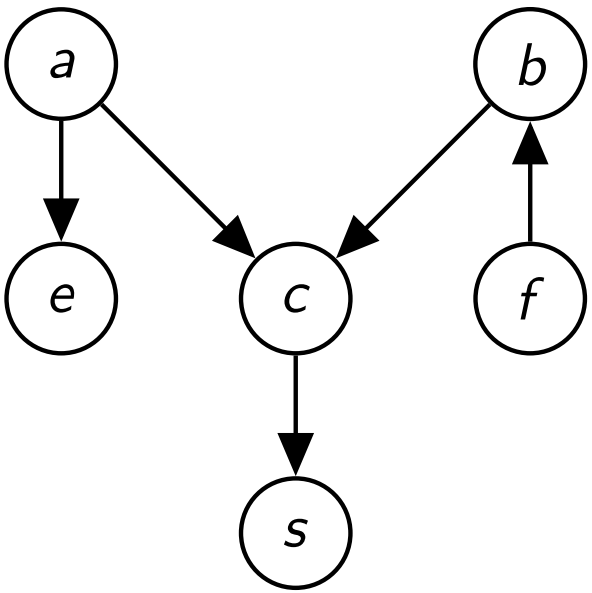
$$z_{i-j} \wedge z_{i-k} \rightarrow z_{j-k} \qquad \{i, j, k\} \cap S = \emptyset \tag{7}$$

$$z_{i-j} \rightarrow \lambda \qquad i \in A, j \in B \tag{8}$$

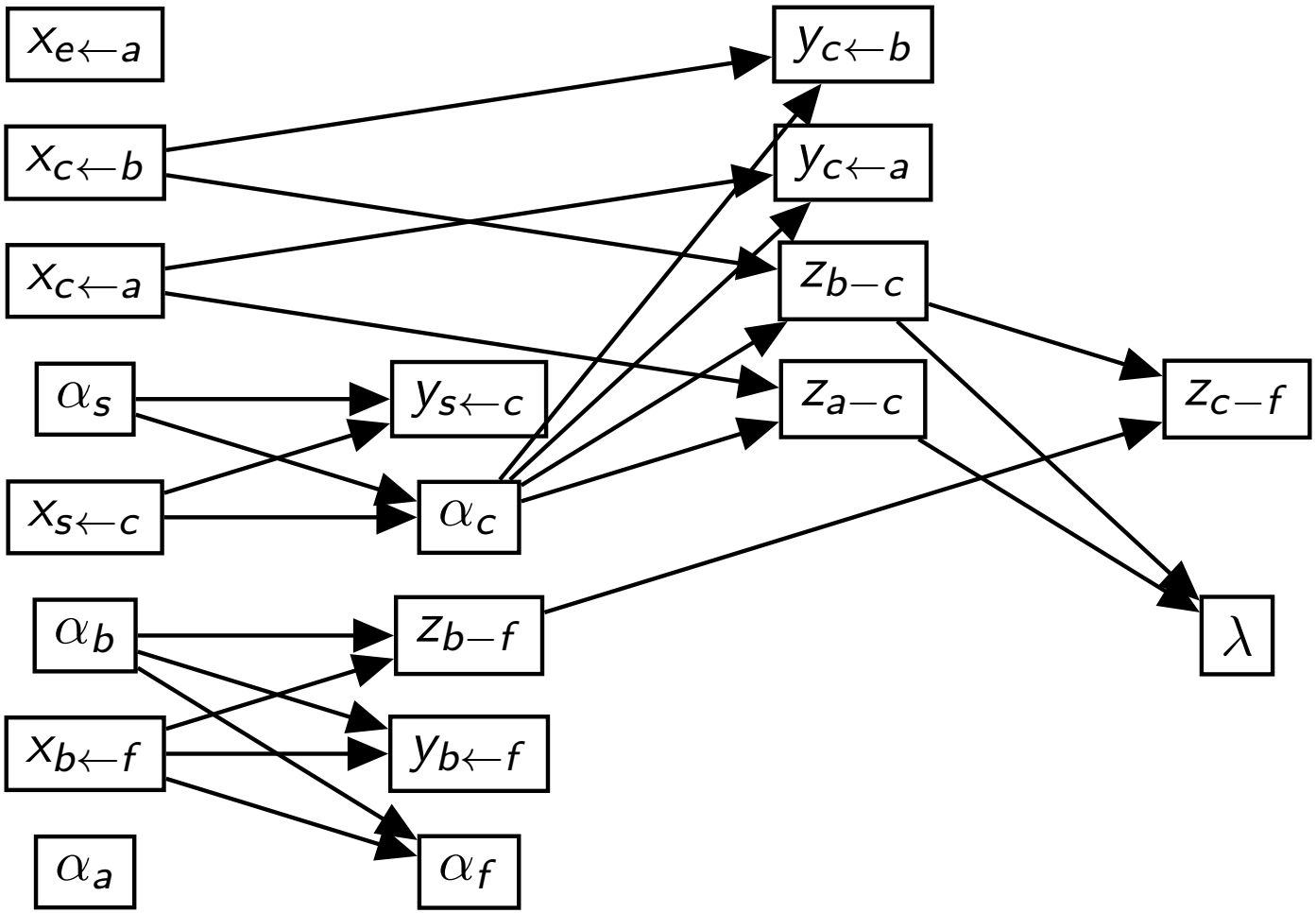
We **do not** create all instances of these rules; a first-order approach is taken instead.

Example graph

This graph violates the constraint $\{a\} \perp \{b\} | \{s\}$:



Example proof (by forward chaining)



CI constraint handling

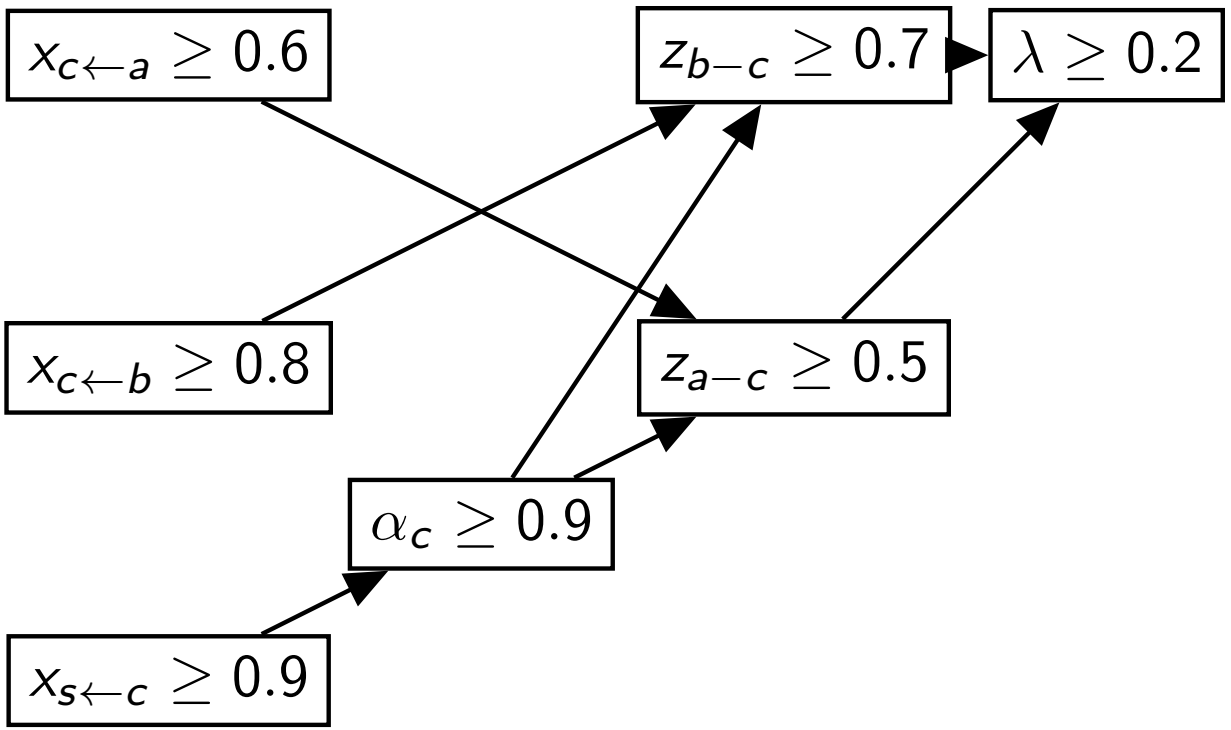
Generalising this logical approach to checking CI constraints we can also implement:

Propagation Given that some arrows are fixed to be present (locally in some node in the branch-and-bound tree) determine which arrows, if any, must be fixed to be absent.

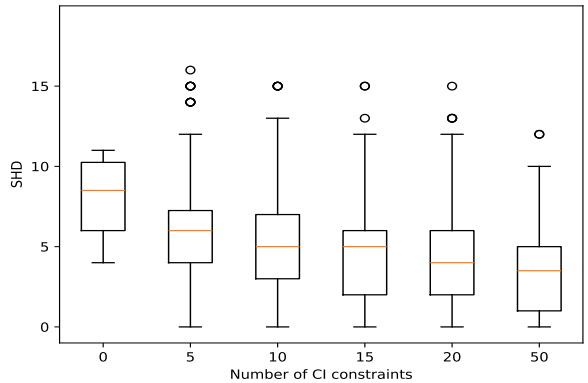
Conflict analysis Record what leads to propagations to infer 'conflict clauses'

Separation Given the solution to a linear relaxation of the problem (or sub-problem) find a linear inequality which follows from a CI constraint which 'cuts off' this solution.

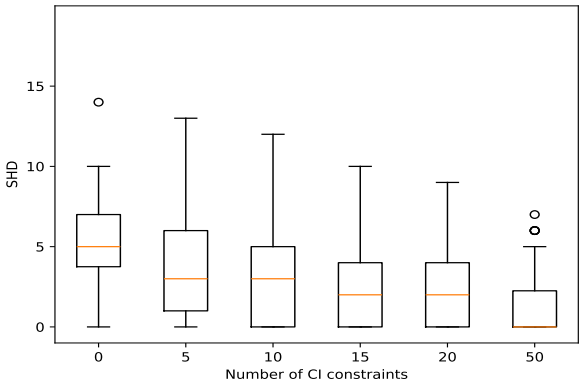
Simplified proof tree for $\lambda \geq 0.2$



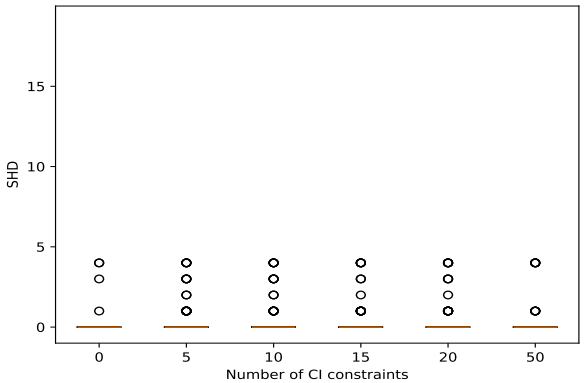
CPDAG SHD with BGe scoring, $p = 10, d = 2$



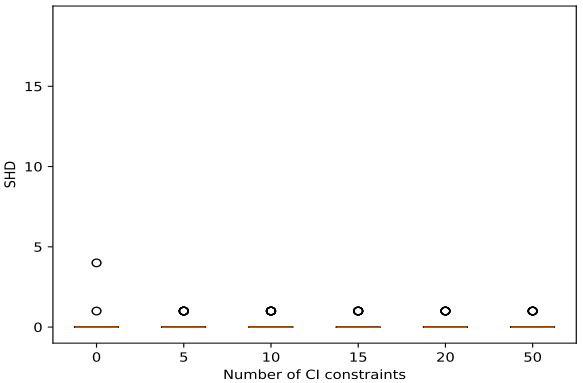
(a) $n = 50$



(b) $n = 100$

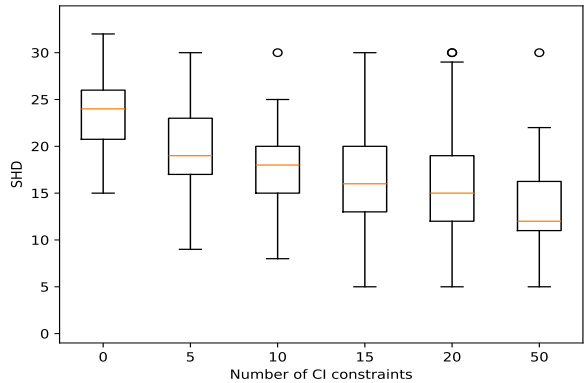


(c) $n = 1000$

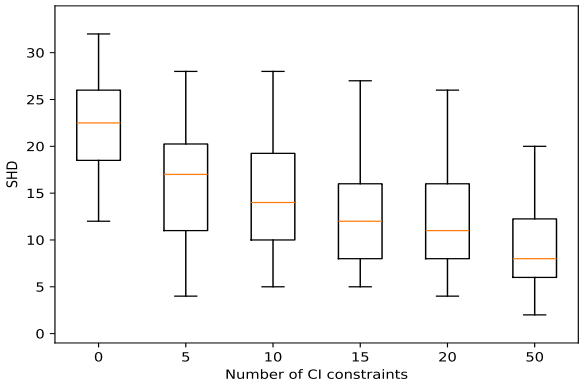


(d) $n = 5000$

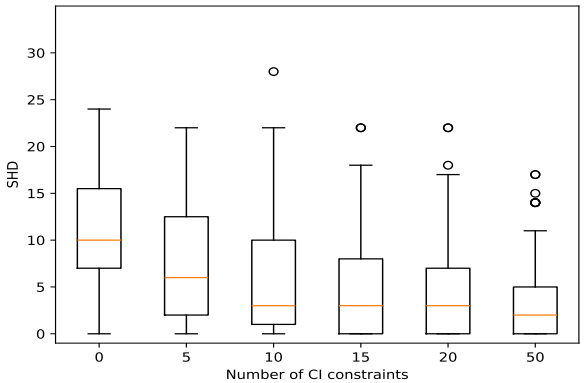
CPDAG SHD with BGe scoring, $p = 10, d = 6$



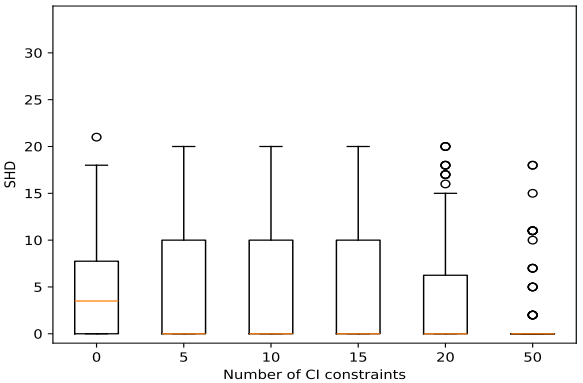
(a) $n = 50$



(b) $n = 100$

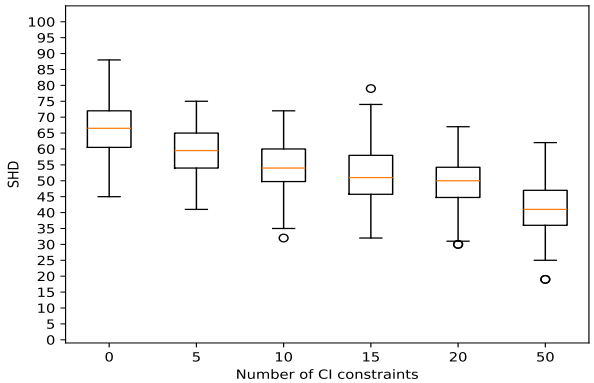


(c) $n = 1000$

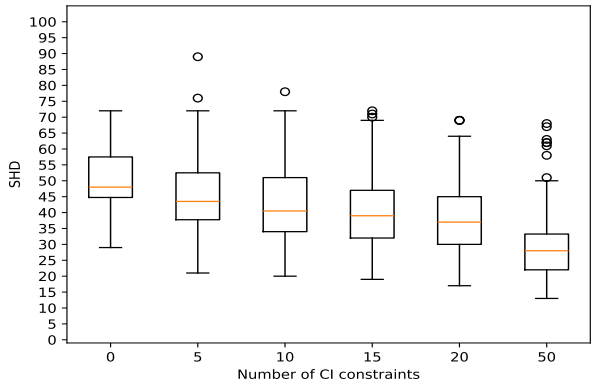


(d) $n = 5000$

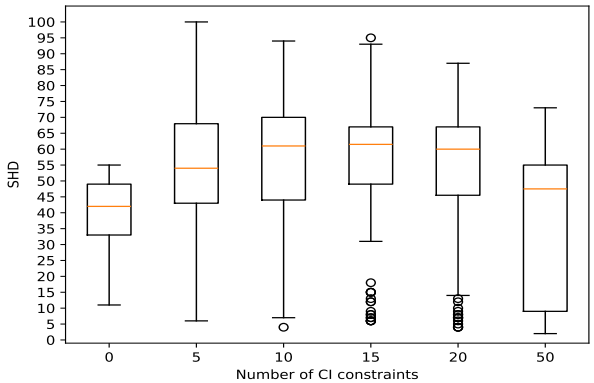
CPDAG SHD with BGe scoring, $p = 20, d = 6, t = 600$



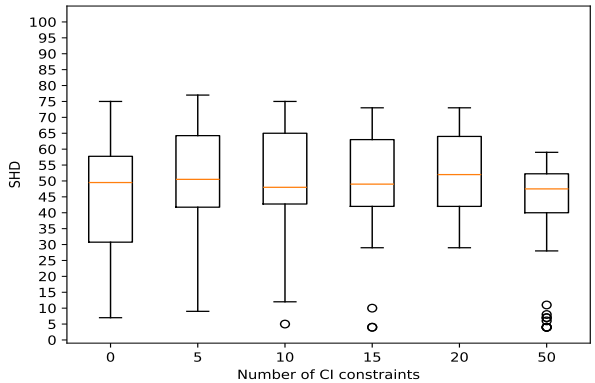
(a) $n = 50$



(b) $n = 100$



(c) $n = 1000$



(d) $n = 5000$

Finding globally optimal solutions

- Assume that we are solving a constrained maximisation problem.
- The *primal bound* is the current best lower bound on the optimal solution.
- The *dual bound* is the current best upper bound on the optimal solution.
- The primal bound is typically provided by the best solution found so far (the *incumbent*). If this primal bound equals the dual bound then we know the incumbent is an optimal solution.

Obtaining dual bounds

- Dual bounds are typically provided by solving a *relaxation* of the original problem, where some constraints are removed.
- A relaxation should be quick to solve and also the resulting optimal objective value should provide a reasonably tight dual bound.
- In integer programming the standard relaxation is the *linear relaxation* where the integrality constraint on integer variables is removed, thus producing a linear program.
- In addition, if there are many (linear) constraints we can further relax by removing most of them. Useful constraints can be added back in during solving as *cutting planes*.

BNSL IP

$$\text{MAXIMISE} \sum_{\substack{i \in P \\ J \subseteq P \setminus \{i\}}} c_{i \leftarrow J} x_{i \leftarrow J}$$

SUBJECT TO:

$$\sum_{J \subseteq P \setminus \{i\}} x_{i \leftarrow J} = 1 \qquad i \in P$$

$$\sum_{i \in C} \sum_{\substack{J \subseteq P \setminus \{i\} \\ J \cap C \neq \emptyset}} x_{i \leftarrow J} \leq |C| - 1 \qquad C \subseteq P, |C| \geq 2$$

$$x_{i \leftarrow J} \in \{0, 1\}, i \in P, J \subseteq P \setminus \{i\}$$

Solving large LPs

- If an LP has too many constraints we can use the cutting plane approach.
- If it has too many variables we can start with just some of the LP variables and use *pricing* to add the ones we need to solve the LP.
- The hope is that most of the full set of LP variables have value 0 in the LP solution, so don't need to be included.

Pricing in action

time	cuts	dualbound	primalbound	gap
p 8.2s	0	--	2.139473e+04	Inf
p 8.8s	0	--	2.691827e+04	Inf
9.5s	0	--	2.691827e+04	Inf
12.8s	0	--	2.691827e+04	Inf
o12.9s	0	--	1.276933e+05	Inf
15.1s	7	--	1.276933e+05	Inf
16.1s	7	--	1.276933e+05	Inf
t16.2s	7	--	1.324030e+05	Inf
17.6s	16	--	1.324030e+05	Inf
18.7s	16	--	1.324030e+05	Inf
19.3s	25	--	1.324030e+05	Inf
19.6s	25	1.498064e+05	1.324030e+05	13.14%
20.1s	34	1.498064e+05	1.324030e+05	13.14%
20.1s	34	1.498041e+05	1.324030e+05	13.14%
o20.2s	34	1.498041e+05	1.345482e+05	11.34%

Pricing problems

- The current pricing algorithm in GOBNILP is a simple generate-and-test search (for variables with negative reduced costs).
- A problem is that the cutting plane approach generates a sequence of LPs, and pricing may be (typically is) required to solve each of them.
- If cutting planes are added at each node of the branch-and-bound tree, then this problem is replicated for each node.
- For bigger problems GOBNILP pricing typically fails in that the linear relaxation is not solved, since we can't be sure we have all the necessary variables (in a reasonable time), and thus we have no dual bound.

Geometric view of the cutting plane approach

- In the 'ideal' formulation of an integer programming problem the linear constraints define the convex hull of feasible solutions.
- But this typically requires too many constraints, so we use a modest number of constraints that define a polytope which contains this convex hull.
- The geometric view of the cutting plane approach is that we add cutting planes to get a polytope closer to the convex hull—at least in the region of optimal vertices.

Dual view of the cutting plane approach

- If $a_1 x \leq b_1$ and $a_2 x \leq b_2$ are two valid inequalities, then $\alpha_1 a_1 x + \alpha_2 a_2 x \leq \alpha_1 b_1 + \alpha_2 b_2$ is also a valid inequality for any $\alpha_1 \geq 0, \alpha_2 \geq 0$.
- If cx is the objective function for an IP and $cx \leq \alpha_1 a_1 x + \alpha_2 a_2 x$ then $\alpha_1 b_1 + \alpha_2 b_2$ is a valid dual bound.
- When we solve the dual of an LP, we are simply looking for the best possible dual bound given the inequalities we have.
- From this perspective, the point of adding a cutting plane inequality is to give the dual LP solver another inequality to use.

A direct dual bound for BNSL

- Rather than adding in lots of inequalities which can be conically combined to give a dual bound, ...
- ...can we not just bound the objective directly?
- Suppose we are looking for a DAG which is optimal for a penalised Gaussian log-likelihood score.
- We know that if the penalty was zero then any complete DAG (the saturated model) would be optimal. Call the objective achieved in that case $L_{\max}$ (which is easy to compute). Let $L(i, J)$ be the local score for $x_{i \leftarrow J}$ when there is no penalty, then:

$$\sum_{\substack{i \in P \\ J \subseteq P \setminus \{i\}}} [L(i, J) - \lambda^2 |J|] x_{i \leftarrow J} \leq \sum_{\substack{i \in P \\ J \subseteq P \setminus \{i\}}} L(i, J) x_{i \leftarrow J} \leq L_{\max}$$

Advantages of the direct dual bound

$$\sum_{\substack{i\in P\\ J\subseteq P\setminus\{i\}}} [L(i,J) - \Lambda^2|J|]x_{i\leftarrow J} \leq \sum_{\substack{i\in P\\ J\subseteq P\setminus\{i\}}} L(i,J)x_{i\leftarrow J} \leq L_{\max}$$

- This gives us a dual bound without the need to create variables $x_{i\leftarrow J}$ or solve any LPs!
- We just add the inequality: $\sum_{\substack{i\in P\\ J\subseteq P\setminus\{i\}}} L(i,J)x_{i\leftarrow J} \leq L_{\max}$, which is valid even if some of the $x_{i\leftarrow J}$ are missing.
- We can also use this in pricing: we take the direct bound into account when computing reduced costs of potential new variables, and ...
- ... if pricing fails we still have the dual bound $L_{\max}$.
- Conically combining with other inequalities can give yet better bounds.

A non-pricing example using the 'direct' bound

time	cuts	dualbound	primalbound	gap
p 1.4s	0	1.336425e+06	-3.528499e+02	Inf
1.8s	0	-1.543534e+02	-3.528499e+02	128.60%
1.9s	4	-1.568406e+02	-3.528499e+02	124.97%
2.0s	15	-1.591819e+02	-3.528499e+02	121.66%
o 2.0s	15	-1.591819e+02	-3.325354e+02	108.90%
....				
o 3.1s	68	-1.617648e+02	-3.173765e+02	96.20%

Optimal solution found after 3.1 seconds.
Problem took 4582s to solve (72k cuts, 16200 search tree nodes)

The same example without using the 'direct' bound

```
 time |cuts |  dualbound   | primalbound  |  gap
p 1.3s|    0 | 1.336425e+06 |-3.528499e+02 |    Inf
  1.6s|    0 | 4.452281e+03 |-3.528499e+02 |    Inf
  2.0s|  124 | 2.594366e+03 |-3.528499e+02 |    Inf
....
L 136s|   17k| 1.530653e+02 |-3.173765e+02 |    Inf
....
 1417s|  251k|-1.430237e+01 |-3.173765e+02 |2119.05%
 1471s|  260k|-2.505175e+01 |-3.173765e+02 |1166.88%
```

Optimal solution found after 136 seconds.
Problem took 2117s to solve (440k cuts, 4697 search tree nodes)

Observations

- *In this particular example*, adding the direct bound allowed a good dual bound very quickly and without many cuts.
- It also lead to the optimal solution being found quickly.
- But proving optimality was a lot slower—the dual bound achievable without the direct bound was able to (eventually) 'catch up'.
- This example provides a good motivation for *concurrent solving*, where several solving strategies are used concurrently.
- SCIP allows the concurrent solves to inform each other of improved primal and dual bounds.

Convex relaxations

- Relaxing to a linear program is not the only option.
- Nonlinear, but convex, relaxations can be quick to solve and may provide a better dual bound.

Gaussian DAG learning with BIC and only arrow variables

$$
\begin{aligned}
\underset{\beta,\omega,a}{\text{MINIMISE}} \quad & \sum_{j\in P}\left[\frac{n}{2}\log(\omega_j^2)+\frac{1}{2\omega_j^2}||x_j - X\beta_j||^2\right] + \lambda\sum_{i\in P, j\in P} a_{ij} \\
\text{SUBJECT TO} \quad & \beta_{ij}\neq 0 \rightarrow a_{ij}=1 && \forall i\in P, \forall j\in P \\
& a \text{ is the adjacency matrix of a DAG} \\
& \beta_{jj}=0 && \forall j\in P \\
& \omega_j^2\in\mathcal{R}^+ && \forall j\in P \\
& \beta_j\in\mathcal{R}^{p\times 1} && \forall j\in P \\
& a_{ij}\in\{0,1\} && \forall i\in P, \forall j\in P
\end{aligned}
$$

A convex relaxation

- Relaxing the constraints in this problem to a set of linear constraints is not too hard.
- However, the objective remains non-convex.
- As Aragam and Zhou [2015] note, Städler et al. [2010] introduced a reparameterisation of the objective which produces a convex function.
- The new parameters are $\rho_j = 1/\omega_j$ and $\phi_{ij} = \beta_{ij}/\omega_j$. Plugging this into the objective produces

$$L(\Phi, R|X) = \sum_{j \in P} \left[-n \log(\rho_j) + \frac{1}{2} ||\rho_j x_j - X\phi_j||^2 \right] \tag{9}$$

where $\Phi = [\phi_1 | \cdots | \phi_p]$ and $R = \text{diag}(\rho_1, \ldots, \rho_p)$. $L(\Phi, R|X)$ is easily seen to be a convex function of (Φ, R).

The convex relaxation

- It is not too hard to reformulate the (relaxed) constraints using the new parameters.
- One can call WORHP ("We Optimize Really Huge Problems") solver from SCIP to solve the convex relaxation.
- Implementational work yet to be done.
- There are interesting questions about whether to integrate this convex relaxation with the standard GOBNILP approach, or whether to solve it as a separate problem.

Finding good solutions where IP variables are missing

- If some $x_{i \leftarrow J}$ variables are missing, then not only is the linear relaxation not directly solvable, ...
- ...it may also make an optimal DAG impossible to represent.
- So in the pricing algorithm, we attempt to improve the incumbent DAG by (greedily) adding arrows to it (which may require the creation of new $x_{i \leftarrow J}$ variables).
- The row with an initial t on slide 20 is an example of this method successfully improving the incumbent.

Bryon Aragam and Qing Zhou. Concave penalized estimation of sparse Gaussian Bayesian networks. *Journal of Machine Learning Research*, 16(69):2273–2328, 2015. URL `http://jmlr.org/papers/v16/aragam15a.html`.

Nicolas Städler, Peter Bühlmann, and Sara Van De Geer. ℓ_1-penalization for mixture regression models. *Test*, 19(2): 209–256, 2010.