# Branch-Price-and-Cut for Causal Discovery

James Cussens, Dept of Computer Science, University of Bristol

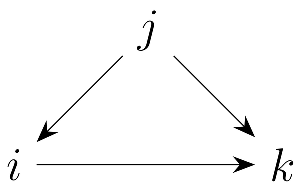
University of BRISTOL

## 1. Casual discovery and optimisation

- There are pros and cons to doing score-based causal discovery (whatever the class of candidate causal models) using a general-purpose constrained optimisation solver.
- Pro: The (highly optimised) software is already there.
- Pro: Can easily add constraints to rule out causal models inconsistent with domain knowledge
- Pro: Exact (sometimes) and anytime (always) learning is possible.
- Con: Need to encode the causal discovery problem in a way the chosen solver understands
- Con: More complex than using a causal-discovery-specific algorithm.

## 2. Encoding DAGs as vectors

This DAG




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

| $x_{i\leftarrow\{\}}$ | $x_{i\leftarrow\{j\}}$ | $x_{i\leftarrow\{k\}}$ | $x_{i\leftarrow\{j,k\}}$ | $x_{j\leftarrow\{\}}$ | $x_{j\leftarrow\{i\}}$ | $x_{j\leftarrow\{k\}}$ | $x_{j\leftarrow\{i,k\}}$ | $x_{k\leftarrow\{\}}$ | $x_{k\leftarrow\{i\}}$ | $x_{k\leftarrow\{j\}}$ | $x_{k\leftarrow\{i,j\}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

- Why this encoding? Because many objective functions ('scores') for DAGs are sums of *local scores* which are determined by the choice of *parents* for each vertex.

## 3. Integer linear programming model for DAG learning

$$\textsc{Min} \sum_{\substack{i\in P\\ J\subseteq P\setminus\{i\}}} c_{i\leftarrow J}\, x_{i\leftarrow J}$$

subject to:

$$\text{(is a directed graph)} \quad \sum_{J\subseteq P\setminus\{i\}} x_{i\leftarrow J} = 1 \qquad i\in P \qquad (1)$$

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

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

- $P$ are the random variables=vertices of the DAG.
- $x_{i\leftarrow J}$ indicates that $J$ is the parent set for child $i$.
- $c_{i\leftarrow J}$ is the local cost (-1 $\times$ local score) when $J$ is the parent set for child $i$.

## 4. Cutting and pricing to solve large linear programs

- The initial step for an ILP solver is to solve the *linear relaxation* of the ILP problem. In our case, this linear relaxation is a linear program (LP) where $x_{i\leftarrow J}\in\{0,1\}$ is replaced $x_{i\leftarrow J}\in[0,1]$.
- There are $2^p - p - 1$ ($p=|P|$) *cluster constraints* ruling out cycles so we add them as *cutting planes*. We initially solve an LP ($\mathrm{LP}_0$) with no cluster constraints to get a solution $x_0^*$ and then add only those cluster constraints that $x_0^*$ violates to get a new LP ($\mathrm{LP}_1$), and then resolve (to get solution $x_1^*$). We repeat until we get an LP ($\mathrm{LP}_m$) whose solution $x_m^*$ satisfies all cluster constraints (even though only a fraction of them are included in the problem).
- But each LP in the sequence $\mathrm{LP}_0, \mathrm{LP}_1, \ldots, \mathrm{LP}_m$ has $p2^{p-1}$ variables!
- For each $\mathrm{LP}_\iota$ we only need to include those $x_{i\leftarrow J}$ variables that have a non-zero value in the solution $x_\iota^*$. (An ILP variable not included in the problem is implicitly set to 0.)
- So, once we have solved $\mathrm{LP}_\iota$ with the ILP variables currently in the problem, we look for additional ILP variables which, if included, would lead to a better (lower cost) solution of $\mathrm{LP}_\iota$. If we can't find any then $\mathrm{LP}_\iota$ has been solved to optimality even though typically a small fraction of ILP variables have been included. This is called *pricing*.

## 5. Pricing

- Associated with each LP solution $x_\iota^*$ there are *dual values* for each constraint in $\mathrm{LP}_\iota$.
- Let $\lambda_i^*$ be the dual value for the constraint (1) for child $i$ and let $\lambda_C^*$ be the dual value for the constraint (2) for cluster $C$.
- A variable $x_{i\leftarrow J}$ is worth adding to the problem if its *reduced cost* $c_{i\leftarrow J} - \lambda_i^* + \sum_{\substack{C\in\mathcal{C}, i\in C\\ C\cap J\neq\emptyset}} \lambda_C^*$ is negative.
- We search for new variables whose reduced cost is minimal:

$$\textsc{Minimise}_J \quad z = c_{i\leftarrow J} - \lambda_i^* + \sum_{\substack{C\in\mathcal{C}, i\in C\\ C\cap J\neq\emptyset}} \lambda_C^*$$

$$\text{subject to} \quad z < 0$$

## 6. Pricing for $\ell_0$ penalised Gaussian DAGs

- For Gaussian DAGs, when the cost is negative log-likelihood with an $\ell_0$ penalty the pricing problem becomes:

$$\textsc{Minimise}_J \quad z = n\log\sigma^2_{i\leftarrow J} + \Lambda^2|J| + \sum_{\substack{C\in\mathcal{C}, i\in C\\ C\cap J\neq\emptyset}} \lambda_C^*$$

$$\text{subject to} \quad z < \lambda_i^*$$

- $n$ is the size of the data. $\Lambda^2$ is the $\ell_0$ penalty.
- $\sigma^2_{i\leftarrow J}$ is the MSE (with MLE parameters) for the linear regression model where variables $J$ predict child $i$.
- Note this is *doubly penalised regression*: we have the normal $\ell_0$ penalty $\Lambda^2|J|$, but also a 'cyclicity' penalty $\sum_{\substack{C\in\mathcal{C}, i\in C\\ C\cap J\neq\emptyset}} \lambda_C^*$

## 7. Optimisations

- If a candidate parent set $J$ for child $i$ has a subset $J'$ with lower cost then it is NOT a *potentially optimal parent set (POP)* and we can rule out $J$ as a parent set for $i$:
$$\forall i\in P, J\subseteq P\setminus\{i\} : \exists J'\subsetneq J : c_{i\leftarrow J}\geq c_{i\leftarrow J'} \rightarrow x_{i\leftarrow J} = 0$$
- Non-POPs should not be priced-in, *even if they have negative reduced costs*.
- We can identify set intervals $\{J : \underline{J}\subseteq J\subseteq \overline{J}\}$ that do not contain POPs and tell the pricing algorithm not to search there.
- Another simpler optimisation is to find all POPs up to some small cardinality before solving begins and tell the pricer to only look for bigger parent sets.
- We can also *delay pricing* so that we only price in new variables once no further cutting planes can be found.
- In one 20 BN variable learning task delayed pricing reduced (exact) solving from 2206 seconds to 351 seconds.

## 8. Implementation and Performance

- GOBNILP has been extended to allow pricing to introduce new ILP variables during solving.
- GOBNILP uses SCIP which (unlike e.g. Gurobi) has support for pricing.
- The pricing problem for the Gaussian log-likelihood $\ell_0$-penalised score is solved by SCIP solving the non-linear optimisation problem (given above).
- If the optimal DAG is sparse then it is often possible to find all POPs before solving starts. This is much faster than pricing them in during solving.
- However, when the optimal DAG is dense, pricing is necessary.
- In one example, produced after the paper was submitted (!), where the optimal DAG had 20 vertices with only one non-adjacency, GOBNILP-with-pricing found it (and proved it optimal) in 140 seconds.
- But standard GOBNILP ran out of memory after 9 minutes.

## 9. Conclusions

More work is required to determine the usefulness and limitations of Branch-Price-and-Cut for Causal Discovery. It would be interesting to apply it to causal discovery beyond DAG learning under causal sufficiency (e.g. to MAG learning).