Learning Optimal Causal Graphs with Exact Search

Kari Rantanen KARI.RANTANEN@HELSINKI.FI
Antti Hyttinen ANTTI.HYTTINEN@HELSINKI.FI
Matti Järvisalo MATTI.JARVISALO@HELSINKI.FI
HIIT, Department of Computer Science, University of Helsinki, Finland

Abstract

Discovering graphical models over very general model spaces with high accuracy requires optimally combining conflicting (in)dependence constraints in sample data, and thus results in a computationally hard combinatorial optimization problem. Recent advances in exact algorithmic approaches in this constraint-based setting build upon off-the-shelf declarative optimization solvers. In this paper, we propose the first truly specialized exact search algorithm for optimal causal graphs in a general model space, allowing both cycles and latent confounding variables. Our problem-oriented approach enables directly incorporating domain knowledge for developing a wider range of specialized search techniques for the problem, including problem-specific propagators, branching heuristics, and bounding techniques, as well as directly incorporating different constraints on the model space, such as sparsity and acyclicity constraints. We empirically evaluate a first implementation of the approach, showing that it clearly outperforms current state of art in exact constraint-based causal discovery on real-world instances.

Keywords: Graphical models; structure learning; causal discovery; exact search; optimization

1. Introduction

Discovering causal relations from sample data when allowing for latent confounding variables and feedback (i.e., cycles) is a very challenging task in the field of graphical models and structure discovery. Although many features of causal structures can in principle be determined even from passive observation (Pearl, 2000; Spirtes et al., 2000), determining which structural features can be identified from finite sample data has proven difficult. For restricted settings without latent confounders and cycles, i.e., for Bayesian networks, exact score-based structure discovery algorithms provide *provably globally optimal graphs* (Yuan and Malone, 2013; Bartlett and Cussens, 2017; van Beek and Hoffmann, 2015) for moderately-sized networks. A central motivation in developing efficient exact algorithms, providing globally optimal results, is that better accuracy can be obtained (Malone et al., 2015). However, much less progress has been made for *exact* discovery algorithms for more general models spaces that allow for latent confounders and cycles.

For more general search spaces (allowing latent confounders), constraint-based causal discovery algorithms (Spirtes et al., 2000; Pearl, 2000) combine (in)dependence constraints from statistical tests to determine common features of the underlying causal graphs. However, such approaches (including PC, CCD and FCI) scale up (wrt number of variables) by selecting independence tests based on earlier test results (Spirtes et al., 2000; Richardson, 1996). Such greedy strategies can lead to non-optimal accuracy in practice, as early mistakes in independence testing guide search towards inaccurate solutions (Claassen and Heskes, 2012; Hyttinen et al., 2014).

Recently it has been shown that better accuracy can be obtained when a predetermined, larger set of tests are conducted before the actual search, and conflicting test results are resolved in an optimal

way via exact methods (Hyttinen et al., 2014; Magliacane et al., 2016; Borboudakis and Tsamardinos, 2016). However, the general model space with latent confounders and cycles induces a combinatorial optimization problem over a drastically larger search space compared to more restricted settings such as Bayesian network structures. Furthermore, the objective functions considered are more complicated to evaluate. Thus improvements to (exact) algorithms for the very general model space in terms of running time performance and scalability without trading off accuracy is a major challenge. While there is recent progress (see Section 5 for more related work) towards improving exact approaches for the general setting, the current state-of-the-art approaches rely heavily on generic off-the-shelf declarative methods such as Boolean satisfiability (SAT) solvers. While declarative methods offer flexibility and remove implementation-level burden of building optimized core search algorithms for the underlying combinatorial optimization tasks, relying on declarative languages for encoding the general model space comes at the price of making it less straightforward to use of key properties of the specific search space within such approaches (Hyttinen et al., 2017).

In this paper we propose a first truly specialized exact search algorithm for optimal causal graphs, allowing both cycles and latent confounding variables. Our problem-oriented view enables directly incorporating domain knowledge for a wider range of specialized search techniques, including problem-specific propagators, branching heuristics, and bounding techniques, as well as directly incorporating restrictions on the model space, such as sparsity and acyclicity constraints. Our direct search over the general model space, implemented as a branch-and-bound approach, also allows for using e.g. linear programming relaxations for obtaining tight bounding during search. We show that a first implementation of the approach compares favourably with current state of the art in exact constraint-based causal discovery on real-world data sets wrt running time performance.

2. Causal Graphs, d-separation and Causal Discovery

We consider the class $\mathcal{G}$ of *causal graphs* $G = (V, E)$ with set of nodes V, where the edge relation E is composed of directed causal edges and (symmetric) bi-directed edges (see Figure 1 for an example). Bidirected edges $X \leftrightarrow Z$ represent *latent confounders*, i.e., structures $X \leftarrow L \rightarrow Z$, where $L \notin V$ is an unmeasured common cause of two observed variables X and Z. When some variables are unobserved, bi-directed edges allow for a canonical representation of causal structures as a graph over the observed variables; Fig. 1 a) can be canonically represented by Fig. 1 b) (Pearl, 2000; Spirtes et al., 2000). We consider both allowing cycles (assuming absence of self-loops as they are unidentifiable from (in)dependence constraints (Lacerda et al., 2008; Hyttinen et al., 2012)) and assuming acyclicity; in the latter case the causal graphs are semi-Markovian graphs (Pearl, 2000).

The central concept connecting statistical dependence to reachability in the causal graph is the following d-separation criterion. A *walk* between X and Y is a sequence of consecutive edges in the graph (allowing for repeated edges and nodes). A node is a *collider* on a walk if both its adjacent edges on the walk point into the node. A walk in graph G is *d-connecting* w.r.t. a conditioning set $C \subseteq V \setminus \{X, Y\}$ if every collider on the walk is in C and no other nodes on the walk are in C. Two nodes are d-connected given a conditioning set C if there is at least one d-connecting walk between them; otherwise they are d-separated. (This is equivalent to Pearl's standard definition (Studený, 1998).) In Fig. 1 b), X and W are d-connected given Y by $X \leftrightarrow Z \rightarrow Y \leftarrow Z \leftarrow T \leftarrow W$. Nodes Y and Q are d-separated given W as all walks between violate the d-connection criterion at X.

Under the commonly used causal Markov assumption (Spirtes et al., 2000), d-separation in the true *acyclic* structure implies statistical independence in the generated distribution. A similar result

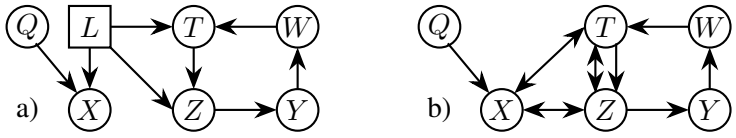

Figure 1: Example graphs: a) a causal graph with an unobserved node L, b) the canonical representation of a) using bidirected edges.

on cyclic causal graphs applies under the following assumptions. The parametric models to cyclic graphs are non-recursive structural equation models (SEMs). We make the standard assumption that each data sample is obtained at the unique solution to the structural equations (given the external disturbances). When the structural equations are linear, d-separation implies independence (Spirtes, 1995). The same result applies for discrete random variables, when every ancestral subset (i.e. the structural equations for a set of nodes and their ancestors) has a unique solution (Forré and Mooij (2017): Thm 3.8.12 on p. 112, see also Pearl and Dechter (1996), Neal (2000)).[1]

Under the commonly used faithfulness assumption (Spirtes et al., 2000), statistical dependence becomes equivalent to (a type of) reachability in the graph: two random variables are statistically dependent conditional on a set of variables C iff they are d-connected given C in the generating causal structure G. That is, given enough samples from a model with structure in Figure 1 b) (or a)), we would expect to find X statistically dependent on W given Y, and Y statistically independent of Q given W. In the rest of the paper we use $X \perp\!\!\!\perp Y|C$ ($X \not\perp\!\!\!\perp Y|C$) to denote statistical independence (dependence) and d-separation (d-connection).

In constraint-based causal discovery, the aim is to find a class of graphs whose d-separation and d-connection properties respectively match the statistical independence and dependence relations in the data. The (in)dependence constraints K are obtained by running statistical independence tests on the data. Since the tests produce some errors on finite sample data, constraint-based causal discovery can be viewed as the following abstract optimization problem (Hyttinen et al., 2014).

INPUT: A set K of conditional (in)dependence constraints over V, and a non-negative weight $w(k)$ for each $k \in K$.

TASK: Find a causal graph $G^* = (V, E^*)$ such that $G^* \in \operatorname{argmin}_{G \in \mathcal{G}} \sum_{k \in K \,:\, G \not\models k} w(k)$.

Our goal is to find a single graph G^* that minimizes the sum of the weights of the (in)dependence constraints *not* implied ($\not\models$) by G^*. The weight function $w(\cdot)$ describes the reliability of each constraint (obtained by independently run tests): conflicts among the constraints are well-resolved when the sum of the weights of the constraints not satisfied is minimized. An optimal graph G^* is a representative of the (Markov) equivalence class closest to the input constraints: there a several ways to examine properties of the class (Hyttinen et al., 2013; Magliacane et al., 2016). Under the assumptions, an exact solution to this formulation retains the consistency and completeness properties of Hyttinen et al. (2013). Weights for the (in)dependence constraints can be obtained by Bayesian model selection (Margaritis and Bromberg, 2009; Claassen and Heskes, 2012) or statistical hypothesis testing (Triantafillou and Tsamardinos, 2015; Magliacane et al., 2016). The algorithm we develop is weight-agnostic. For our experiments, we obtain weights from simple local Bayesian model selection (Hyttinen et al., 2014).

1. Forré and Mooij (2017, 2018) also formulate a different separation condition for non-linear cyclic models. Our search procedure can be modified to use this criterion instead.

3. Branch and Bound for Causal Discovery

In this section we describe a first specialized branch-and-bound approach to finding optimal causal graphs. After an overview we give details on how to efficiently evaluate the objective function under partial solutions and thereby infer further knowledge of guaranteed dependencies and independencies under partial solutions (Sect. 3.1), an effective domain-specific branching heuristic (Sect. 3.2), and how to obtain tight bounds initially (Sect. 3.3) and during search using linear programming relaxations (Sect. 3.4). Furthermore, we describe how structural restrictions on the model space, such as enforcing acyclicity and degree-restrictions (Sect. 3.5), can be integrated.

The overall structure of the branch-and-bound search is shown in Algorithm 1. A *partial solution* G is a graph in which each edge is either decided *absent*, decided *present* or *undecided*; see Figure 2 a) for an example. At each search tree node, on Line 2 we estimate a lower bound for the weight of partial solution G. If this value is not less than the weight of the incumbent upper bound solution G^*, we can safely close the current branch and bracktrack. If the current branch cannot be closed, we move on to Line 3 to select a yet-undetermined edge e^* in G. If no such edge exists, i.e., $e^* = null$, we update the incumbent upper bound solution G^* to G if the current partial solution has smaller weight. If multiple edge candidates exist, the most promising one is chosen via a selected heuristic. On the other hand if $e^* \neq null$, i.e., a decidable edge exists, we recursively call Algorithm 1 to open two search tree branches, one where (a) e^* is decided present in G and one where (b) the edge is decided absent. The order in which we visit these branches is determined heuristically, see Section 3.2 for details. At the end of the search, G^* is guaranteed to be a solution with globally optimal cost.

3.1 Efficient Evaluation of the Objective Function

Given that there are superpolynomial number of (in)dependence constraints with respect to the number of graph nodes, evaluating the objective function can be a time-consuming task in itself. In this section we provide ideas for efficient, incremental tracking of satisfiability for given constraints. There are several different ways for checking whether (in)dependence constraints are satisfied by a graph (Studený, 1998; Shachter, 1998). Building on such ideas, here our focus is to evaluate a large number of constraints incrementally when going down a branch, and the constraints are evaluated over a partial solution, a graph for which some edges are decided present and some absent.

For a partial solution, each (in)dependence constraint can have exactly one of three states: *satisfied*, *violated* or *undetermined*. The states are defined in the following way. A *complete solution* or *completion* can be obtained from a partial solution by deciding the state of all undecided edges. A *maximal completion* $\mathrm{maxc}(G)$ of a partial solution has all undecided edges marked present (e.g.

Algorithm 1 The core structure of the branch and bound.

```
1: function SEARCH(Partial solution G)
2:     if w(G*) ≤ LOWERBOUND(G) then return
3:     e* ← SELECTUNDECIDEDEDGE(G)
4:     if e* ≠ null then
5:         Branch with (a) SEARCH(G with e* decided present) and
6:             (b) SEARCH(G with e* decided absent) in the preferred order.
7:     else if w(G) < w(G*) then  G* ← G
```

Figure 2 a) *with* the dashed edges), a *minimal completion* $\mathrm{minc}(G)$ has all undecided edges absent (Figure 2 a) *without* the dashed edges). An independence constraint is satisfied if the corresponding d-separation holds in $\mathrm{maxc}(G)$, and violated if the corresponding d-separation does not hold in $\mathrm{minc}(G)$. A dependence constraint is satisfied if the corresponding d-connection holds in $\mathrm{minc}(G)$, and violated if the corresponding d-connection does not hold in $\mathrm{maxc}(G)$. All other constraints are undetermined. For the partial solution in Figure 2a, constraint $[X \not\perp\!\!\!\perp Z \mid W]$ is satisfied, $[X \perp\!\!\!\perp T \mid W]$ is violated and constraint $[X \not\perp\!\!\!\perp T]$ remains undetermined. In the beginning of the search, when no edges are decided, the states of all constraints are undetermined.

When a new edge decision is made, we update the states of the input constraints with respect to the current partial solution. This also keeps track of the total weight of the violated constraints and provides a simple lower bound. Furthermore, the satisfied/violated information can be given to a linear programming solver so that the stronger, core-based lower bounds stay up to date (Section 3.4). Note that the choice of how regularly we keep the constraint states up to date does not affect the correctness of the search, as long as all complete solutions are evaluated exactly.

At each search node, we branch on a currently undecided edge to be either present or absent in the partial solution. When deciding an edge present, assuming that acyclity or an edge degree limit is not enforced, we only check whether new d-connections are formed in the minimal completion of the partial solution. When deciding an edge absent, we only check whether d-connections disappeared from the maximal completion of the partial solution.

An efficient way to update constraint states for a given node pair after an edge decision is proposed in Algorithm 2. For example, consider a case where an edge $A \to B$ is decided present in a partial solution G. To update constraint states for a node pair (X, Y), we first check whether there could be a new d-connection between X and Y in the minimal completion G' (Line 3). If not, the constraints need not be updated. Otherwise, we identify a set of unavoidable colliders C^+ and non-colliders C^- on any d-connecting walk from X to Y in G' (Line 4 and 5). We can omit checking any constraint states for X and Y where conditional set does not contain all the colliders C^+ or contains some non-colliders C^-. Each item in these sets halves the number of constraints that we need to check for the node pair in question. Intuitively, for any relatively sparse partial solution, there is likely a shared bottleneck for all walks between two nodes.

There is no need for the C^+/C^- sets to contain every single unavoidable collider/non-collider and hence we need to make a tradeoff between how much time is used to gather C^+/C^- and how much time is saved by having those sets. For this reason we use the following straightforward method for gathering only some of the unavoidable colliders/noncolliders when traversing from node X to Y in the (minimal/maximal) completion G' of partial solution G. Let node $A \in \{X, Y\}$ be a starting point and let $V = \emptyset$ be the set of visited nodes within the procedure.

1. Let $nb(A)$ be the neighbours of A in G'.

Algorithm 2 An efficient method for updating constraint states after an edge decision.

1: **function** CHECKCONSTRAINTS(Partial solution G, Node pair (X, Y), Edge e)
2: **if** e is present in G **then** $G' \leftarrow \mathrm{minc}(G)$ **else** $G' \leftarrow \mathrm{maxc}(G)$with$e$
3: **if** e does not affect the d-connectivity of X and Y in G' **then return**
4: $C^+ \leftarrow$ Unavoidable colliders on d-connections between X and Y in G'.
5: $C^- \leftarrow$ Unavoidable non-colliders on d-connections between X and Y in G'.
6: Check constraints $[X \perp\!\!\!\perp Y \mid Z]$ (corr. $[X \not\perp\!\!\!\perp Y \mid Z]$) s.t. $C^+ \subseteq Z$ and $Z \cap C^- = \emptyset$

2. If $A \notin \{X, Y\}$ and for all $N \in nb(A)$ the edge $A \to N$ is not present in G', update $C^+ \leftarrow C^+ \cup \{A\}$.
3. If $A \notin \{X, Y\}$ and for all $N \in nb(A) \setminus V$ neither $A \leftarrow N$ nor $A \leftrightarrow N$ is present in G', update $C^- \leftarrow C^- \cup \{A\}$.
4. If $|nb(A) \setminus V| \neq 1$ or $nb(a) \setminus V \subseteq \{X, Y\}$, terminate the procedure.
5. Let $B \in nb(A) \setminus V$. If neither $A \to B$ nor $A \leftrightarrow B$ is present in G', update $C^- \leftarrow C^- \cup \{B\}$.
6. Assign $V \leftarrow V \cup \{A\}$ and $A \leftarrow B$, and return to step 1.

We execute this procedure starting from both X and Y. For example, for the partial solution in Figure 2 a), we find that node Z is an unavoidable non-collider in all d-connecting walks between X and Y (given any conditioning set) in the corresponding minimal completion.

3.2 Problem-Specific Branching Heuristics

The branching heuristics applied within the branch and bound are crucial for the performance of the algorithm. In this section we propose problem-specific heuristics for our approach.

Let $K^+(X, Y)$ ($K^-(X, Y)$) be the set of undetermined dependence (independence) constraints between nodes (X, Y) by the current partial solution. We will also use $K^+(X) = \bigcup_Y K^+(X, Y)$ ($K^-(X) = \bigcup_Y K^-(X, Y)$) to denote the undetermined dependence (independence) constraints involving node X. Furthermore, let $w^+(X, Y), w^-(X, Y), w^+(X), w^-(X)$ be the sum of weights of the constraints in sets $K^+(X, Y), K^-(X, Y), K^+(X), K^-(X)$, respectively. We use the following rules for choosing the next pair for which an edge to be decided absent or present. Here (X, Y) and (A, B) denote distinct pairs of nodes.

1. Choose (X, Y) over (A, B) if all edges are decided between (A, B) or more edges have been decided present between (A, B) than between (X, Y).
2. Choose (X, Y) over (A, B) if $w^-(A, B) \leq w^-(X, Y)$ and $w^-(X, Y) > 0$.
3. Choose (X, Y) over (A, B) if $w^+(X) + w^+(Y) + \max_{k \in K^+(X,Y)} w(k) \geq w^+(A) + w^+(B) + \max_{k \in K^+(A,B)} w(k)$.

The first rule captures our preference of setting edge decisions throughout the entire graph instead of deciding all edges between a single pair of nodes immediately. The second rule captures the preference for edges absences when the involved nodes are found independent given one or many conditioning sets. Deciding these absences of edges early via the heuristic directs the search towards sparser solutions for which d-connection checks are faster to evaluate. This relates to previous literature: PC algorithm decides the absence of an edge between X, Y upon finding a single conditioning set given which the nodes are independent (Spirtes et al., 2000). Thus, a problem-specific greedy (and often unreliable) strategy can act as a good heuristic in exact search. Finally via the third rule we prefer satisfying strong dependencies with large weights using direct connections.

After the best node pair (X, Y) is chosen out of the possible options, we branch in the search by deciding an arbitrary yet-undecided edge between the nodes ($X \to Y$, $X \leftarrow Y$ or $X \leftrightarrow Y$). We branch by deciding the edge absent first iff $[X \perp\!\!\!\perp Y \mid Z] \in K$ for any $Z \subseteq \mathcal{V} \setminus \{X, Y\}$.

3.3 Obtaining an Initial Solution as an Upper Bound

Algorithm 3 describes a simple method for computing an initial upper bound solution G^*. We start with empty graphs G' and G^*. We traverse dependence constraints $[X \not\perp\!\!\!\perp Y \mid Z] \in K$ in descending weight order (Triantafillou et al., 2010) and add corresponding edges $X \to Y$ to G' (unless this would violate possible acyclicity or degree bounds; see Section 3.5). If at any point

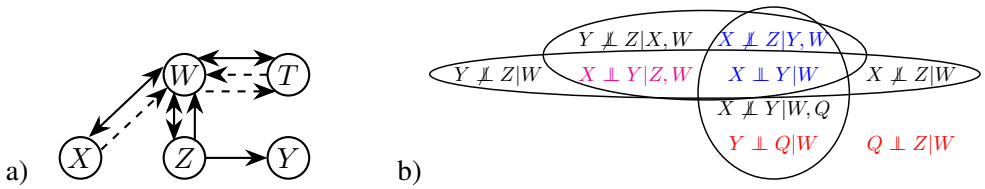

Figure 2: Left: A partial solution; regular edges have been decided present, dashed edges remain undecided, others are decided absent. Right: Example of core-based lower bounding.

G' has lower weight than G^*, G^* is updated to G'. While there are a number of ways to obtain an initial solution (any method, e.g. FCI (Spirtes et al., 2000) would do), we found this procedure fast and effective in finding a good bound, and it also integrates directly to our search procedure.

3.4 Computing Tight Lower Bounds by Linear Programming

We now describe how we compute strong lower bounds using core patterns (Hyttinen et al., 2017). An *unsatisfiable core* is a set of (in)dependence constraints that cannot be not simultaneously satisfied by any graph in $\mathcal{G}$. Some example cores are marked by ellipses in Figure 2 b). We use the seven core patterns from (Hyttinen et al., 2017) to find cores for the input dataset in the beginning of the search. Using these, we can compute lower bounds by formulating a minimum-cost hitting set problem where the unsatisfiability cores represent the sets and the (in)dependence constraints represent the elements. The objective is then to find a minimum-cost subset of constraints that contains something from each core. To obtain the bounds in practice, we solve linear relaxations of a standard integer programming formulations of these hitting set problems using a linear programming (LP) solver, similarly as Hyttinen et al. (2017). Furthermore, as we update the constraint states (Section 3.1) for the current partial solution, we can adapt our hitting set computation to take this information into consideration. For example, if some constraint is known to be satisfied in the partial solution, we are not allowed to choose it into our hitting set. Conversely, a hitting set has to contain all the constraints that are known to be unsatisfied for the partial solution. This way the core-based lower bounds are constantly updated to match the current search tree branch.

As an example, suppose we have the cores in Figure 2 b) and the partial solution satisfies $X \not\perp\!\!\!\perp Z|Y,W$ and $X \perp\!\!\!\perp Y|W$ (in blue), and violates $Y \perp\!\!\!\perp Q|W$ and $Q \perp\!\!\!\perp Z|W$ (in red). One constraint in each core marked by ellipses must be chosen. The violated constraint $Y \perp\!\!\!\perp Q|W$ hits the core

Algorithm 3 A naive algorithm for computing an initial UB solution.

function NAIVEUB
  Initialize empty graphs G^* and G'.
  $K' \leftarrow \{[X \not\perp\!\!\!\perp Y \mid Z] \in K\}$
  while $K' \neq \emptyset$ **do**
    Select $[X \not\perp\!\!\!\perp Y \mid Z] \in \arg\max_{k \in K'} w(k)$.
    if $X \rightarrow Y$ can be added to G' without violating acyclicity/sparsity constraints **then**
      $G' \leftarrow G'$ with edge $X \rightarrow Y$ added.
      if $w(G^*) > w(G')$ **then** $G^* \leftarrow G'$
    $K' \leftarrow K' \setminus \{[X \not\perp\!\!\!\perp Y \mid Z]\}$
  return G^*

marked by the vertical ellipse. If for simplicity the weights are all constants, the remaining cores can be optimally hit by $X \perp\!\!\!\perp Y|Z, W$ (in magenta). Thus, the final lower bound for the partial solution is $w(Y \perp\!\!\!\perp Q|W) + w(Q \perp\!\!\!\perp Z|W) + w(X \perp\!\!\!\perp Y|Z, W)$, where the last term in the sum tightens the bound compared to the simple bound due to just violated constraints.

3.5 Imposing Acyclicity and Sparsity

Our approach also allows for integrating different structural model space restrictions. We now explain how to enforce two types of constraints: acyclicity and sparsity.

To enforce acyclicity (in terms of directed edges), we keep track of the set $R[X]$ of nodes reachable by a directed path of decided edges from node X in the current partial solution. Initially $R[X] = \emptyset$ for each node X. After an edge $X \rightarrow Y$ is decided present, we update $R[Z] \leftarrow R[Z] \cup \{Y\} \cup R[Y]$ for each $Z \in \{X\} \cup \{Z' : X \in R[Z']\}$. Using this information, we can decide any edge $X \rightarrow Y$ as absent in all completions of the current partial solution where $X \in R[Y]$.

We can also enforce sparsity constraints, such as a bound on the maximum degree of nodes (as used by Claassen et al. (2013)), in a straightforward way. We can simply keep track of the degree for each node in the current partial solution, and decide all the yet-undecided edges between a node pair to be absent if the degree for either node has already reached the maximum allowed value.

4. Empirical Evaluation

We report on an empirical comparison between a first implementation *bcause* of the branch-and-bound approach detailed in Section 3, and the recently proposed approach *dseptor* (Hyttinen et al., 2017) representing the current state of the art. We use CPLEX as the linear programming solver for obtaining core-based bounds. The benchmark instances were generated from real-world datasets often used for benchmarking exact Bayesian network structure learning algorithms (Yuan and Malone, 2013; Bartlett and Cussens, 2017); see Table 1. We considered suitable-sized (n) subsets of the first non-constant variables in the datasets, the remaining variables becoming thus latent (latent variables are supported by our general model space), resulting in a total of 119 benchmark instances. The data is discrete, so the constraint weights are obtained by (local) Bayesian model selection with the BDeu (ESS=10) score. The experiments were run under a 1-h per-instance time limit on Intel Xeon E5-2680 v4 2.4GHz processors and 256-GB RAM.

The results are shown in Figure 3 and Table 1, with comparisons of bcause and dseptor both without structural restrictions on the model space (Fig. 3 middle) and when enforcing acyclicity (Fig. 3 right). The plot in Fig. 3 left gives the number of instances solved (x-axis) by each approach under different per-instance time limits (y-axis); the further to the right the line, the better. On a clear majority of the benchmarks bcause exhibits noticeably better runtimes than dseptor regardless of whether acyclicity is enforced, and times out less frequently, with 7 and 46 timeouts, respectively, without enforcing acyclicity, 7 and 50 timeouts under acyclicity. Table 1 gives per-instance details for largest n instances, with the time to reach an optimal solution (without yet proving optimality) shown for bcause in parentheses. Furthermore, the Max-degree 3 column gives runtimes for bcause when enforcing that the maximum node degree is at most three. We also observe that bcause exhibits very good anytime performance in that it reaches an optimal solution often relatively fast.

Table 1: Running times of bcause and dseptor over different model spaces.

		Running times (s)							
		General			Acyclic			Max-degree 3	
Dataset	n	**bcause**		**dseptor**	**bcause**		**dseptor**	**bcause**	
Adult	7	**2837**	(780)	>3600	**1773**	(234)	>3600	1535	(178)
Alarm	9	**540**	(6)	>3600	**532**	(7)	>3600	579	(7)
Autos	9	**519**	(7)	>3600	**505**	(6)	>3600	502	(7)
Bands	8	**364**	(3)	>3600	**291**	(2)	>3600	367	(3)
Epigenetics	7	>3600		**16**	>3600		**144**	>3600	
Flag	12	**209**	(158)	>3600	**235**	(188)	>3600	225	(173)
Heart	12	**160**	(29)	2290	**190**	(27)	2287	176	(20)
Hepatitis	12	**225**	(32)	>3600	**271**	(30)	>3600	241	(21)
Horse.23	9	**155**	(9)	>3600	**161**	(8)	>3600	173	(10)
Horse	11	**51**	(8)	>3600	**60**	(8)	>3600	57	(6)
Image	8	**949**	(30)	>3600	**2487**	(57)	>3600	1041	(72)
Imports	8	**335**	(2)	>3600	**308**	(2)	>3600	342	(2)
Letter	7	>3600		**433**	>3600		**549**	>3600	
LungCancer	10	**692**	(24)	>3600	**835**	(27)	>3600	794	(26)
Meta	7	>3600		**112**	>3600		**132**	>3600	
Mushroom1000	7	>3600		**1187**	>3600		**767**	>3600	
Mushroom8124	7	>3600		**662**	>3600		**800**	>3600	
Parkinsons	7	**500**	(33)	>3600	**264**	(10)	>3600	217	(14)
Sensors	7	>3600		**62**	>3600		**144**	>3600	
Soybean	11	**44**	(9)	983	**53**	(9)	930	47	(6)
Spectf	11	**53**	(9)	798	**63**	(9)	818	55	(5)
Statlog	7	46	(18)	**32**	**377**	(35)	>3600	39	(14)
SteelPlates	6	618	(566)	**56**	157	(115)	**31**	62	(23)
Voting	9	**900**	(12)	>3600	**942**	(12)	>3600	968	(13)
Water	9	**589**	(17)	>3600	**837**	(4)	>3600	615	(11)
Wdbc	8	**75**	(50)	2997	**52**	(30)	2779	62	(38)
Wine	9	**1316**	(15)	>3600	**1242**	(15)	>3600	1387	(17)
Zoo	7	**244**	(112)	>3600	**142**	(43)	>3600	222	(83)
alarm10000	9	28	(28)	**14**	**11**	(11)	14	5	(5)
alarm1000	10	**203**	(57)	>3600	**184**	(35)	>3600	185	(23)
alarm100	9	**28**	(4)	389	**30**	(5)	406	31	(4)
asia10000	8	**177**	(88)	276	**151**	(51)	328	324	(112)
asia1000	7	154	(<1)	**41**	82	(<1)	**31**	136	(35)
asia100	7	4	(<1)	**2**	2	(<1)	2	3	(<1)
carpo10000	11	**212**	(121)	2727	**151**	(100)	2613	140	(87)
carpo1000	10	**2258**	(3)	>3600	**1999**	(3)	>3600	2277	(2)
carpo100	10	**65**	(2)	>3600	**62**	(2)	>3600	67	(2)
Diabetes10000	9	>3600		**748**	>3600		**1210**	>3600	
Diabetes1000	8	**<1**	(<1)	6	**1**	(<1)	6	<1	(<1)
Diabetes100	9	**897**	(<1)	>3600	**1141**	(<1)	>3600	1031	(<1)
hailfinder10000	10	**13**	(2)	163	**16**	(2)	169	14	(1)
hailfinder1000	9	**3**	(<1)	63	**3**	(<1)	36	3	(<1)
hailfinder100	8	**2084**	(<1)	>3600	**2546**	(<1)	>3600	2326	(<1)
insurance10000	9	**398**	(4)	>3600	**401**	(4)	>3600	428	(4)
insurance100	10	**1053**	(14)	>3600	**1148**	(17)	>3600	1105	(15)
Link10000	12	**353**	(19)	>3600	**348**	(19)	>3600	337	(18)
Link1000	10	**22**	(2)	207	**24**	(2)	225	22	(1)
Link100	10	**974**	(13)	>3600	**1135**	(16)	>3600	1082	(14)
Mildew10000	10	**14**	(13)	527	**16**	(15)	595	15	(13)
Mildew1000	8	**4**	(<1)	36	**5**	(<1)	37	4	(<1)
Mildew100	6	**<1**	(<1)	8	**<1**	(<1)	13	<1	(<1)
Pigs10000	10	**561**	(1)	>3600	**634**	(1)	>3600	668	(1)
Pigs1000	10	**16**	(11)	3544	**17**	(12)	>3600	17	(12)
Pigs100	8	**1**	(<1)	24	**2**	(1)	20	1	(1)
Water10000	9	**98**	(4)	>3600	**95**	(4)	>3600	107	(5)
Water1000	12	**122**	(120)	>3600	**135**	(133)	>3600	142	(139)
Water100	11	**59**	(39)	>3600	**67**	(46)	>3600	64	(42)

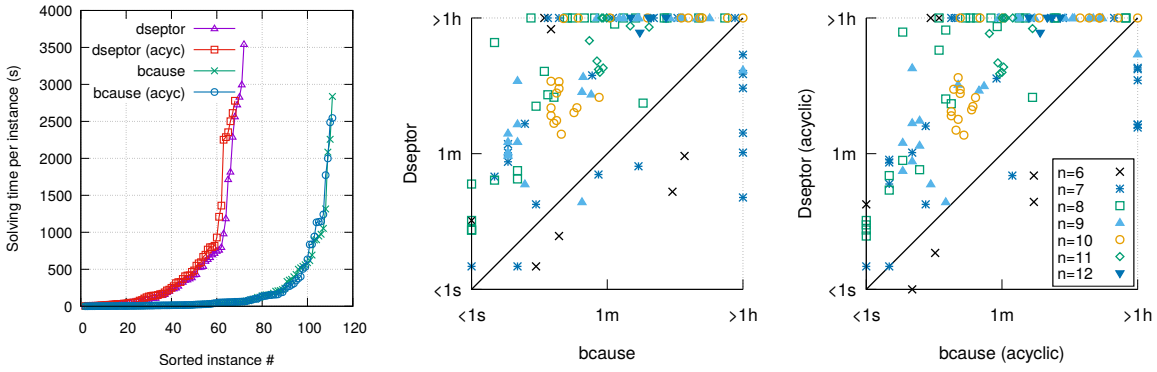

Figure 3: Running time performance comparison: bcause v dseptor.

5. Related Work

There has been noticeable interest in developing algorithmic solutions to problem settings related to ours. Declarative solvers, specifically Boolean satisfiability (SAT) solvers, were first used in (Triantafillou et al., 2010; Triantafillou and Tsamardinos, 2015; Hyttinen et al., 2013) for developing approaches to discovering causal structures from experimental data sets. The first exact approach to the problem we focus on here was proposed in (Hyttinen et al., 2014), where a declarative framework in the language of answer set programming (ASP) was proposed to obtain optimal solutions. Relaxing requirements of optimal solutions, this framework was subsequently adapted to formulate a relaxed version with focus on several experimental data sets (Magliacane et al., 2016) and to examine different types of relaxed faithfulness conditions (Zhalama et al., 2017). Furthermore, a different encoding was proposed in (Borboudakis and Tsamardinos, 2016). Up until now, the current state of the art to the exact problem setting we consider here is the recent Maximum satisfiability (MaxSAT) based approach developed in (Hyttinen et al., 2017), where specific domain-specific techniques were integrated to the extent possible to a MaxSAT solver, still relying on a MaxSAT solver to solve the search problem starting with a declarative encoding of the problem.

Beyond exact and declarative approaches, several domain-specific algorithms have been proposed for discovering causal models with cycles or latent confounders: FCI learns acyclic graphs allowing for latent confounding and selection bias (Spirtes et al., 2000), CCD learns graphs with cycles (Richardson, 1996). FCI was further developed later by (Colombo et al., 2012) for efficiency. Claassen and Heskes (2012) combine weighted (in)dependence constraints with a similar procedure. A polynomial-time algorithm for discovering edge-minimal acyclic graphs with latent variables was recently developed Claassen et al. (2013). Further related work in score-based learning includes Tsirlis et al. (2017) and Evans and Richardson (2010) who score and learn causal graphs with latent confounding assuming linear Gaussianity or binary variables respectively.

Finally, we note that different branch and bound style algorithms have also been proposed for other structure learning tasks (Suzuki, 1996; Tian, 2000; Malone and Yuan, 2014; van Beek and Hoffmann, 2015; Suzuki and Kawahara, 2017; Rantanen et al., 2017).

6. Conclusions

We proposed a first truly specialized exact algorithm for the problem of finding optimal causal graphs in a general model space. A key benefit of our approach is direct integration of domain

knowledge for developing specialized search techniques for the problem. To this end, we described directly implementable branching heuristics, evaluation and propagation techniques, as well as specialized and LP-based bounding techniques. Further, we explained how model space restrictions such as acyclicity and sparsity—without having to resort to declaratively enforcing acyclicity—can be integrated. A first implementation of the approach exhibits better running time performance than current state of art in exact constraint-based causal discovery on real-world data sets. We foresee various direction for further work. For example, the approach allows for extensions towards different separation criterions, to allowing selection bias, and to the use of multiple experimental data sets. For runtime improvements, the approach currently does not make use of problem-specific symmetries, which is also a non-trivial yet promising direction for further work.

Acknowledgments

Work supported by Academy of Finland (grants 276412, 295673, 312662) and the Research Funds of the University of Helsinki.

References

M. Bartlett and J. Cussens. Integer linear programming for the Bayesian network structure learning problem. *Artif. Intell.*, 244:258–271, 2017.

G. Borboudakis and I. Tsamardinos. Towards robust and versatile causal discovery for business applications. In *KDD*, pages 1435–1444. ACM, 2016.

T. Claassen and T. Heskes. A Bayesian approach to constraint based causal inference. In *UAI*, pages 207–216. AUAI Press, 2012.

T. Claassen, J. M. Mooij, and T. Heskes. Learning sparse causal models is not NP-hard. In *UAI*, pages 172–181. AUAI Press, 2013.

D. Colombo, M. H. Maathuis, M. Kalisch, and T. S. Richardson. Learning high-dimensional directed acyclic graphs with latent and selection variables. *Ann. Statist.*, 40(1):294–321, 2012.

R. J. Evans and T. S. Richardson. Maximum likelihood fitting of acyclic directed mixed graphs to binary data. In *UAI*, pages 177–184. AUAI Press, 2010.

P. Forré and J. M. Mooij. Markov properties for graphical models with cycles and latent variables. *arXiv.org preprint*, arXiv:1710.08775 [math.ST], 2017.

P. Forré and J. M. Mooij. Constraint-based causal discovery for non-linear structural causal models with cycles and latent confounders. In *Proc. UAI*, 2018.

A. Hyttinen, F. Eberhardt, and P. O. Hoyer. Learning linear cyclic causal models with latent variables. *J. Mach. Learn. Res.*, 13:3387–3439, 2012.

A. Hyttinen, P. O. Hoyer, F. Eberhardt, and M. Järvisalo. Discovering cyclic causal models with latent variables: A general SAT-based procedure. In *UAI*, pages 301–310. AUAI Press, 2013.

A. Hyttinen, F. Eberhardt, and M. Järvisalo. Constraint-based causal discovery: Conflict resolution with answer set programming. In *UAI*, pages 340–349. AUAI Press, 2014.

A. Hyttinen, P. Saikko, and M. Järvisalo. A core-guided approach to learning optimal causal graphs. In *IJCAI*, pages 645–651. ijcai.org, 2017.

G. Lacerda, P. Spirtes, J. Ramsey, and P. O. Hoyer. Discovering cyclic causal models by independent components analysis. In *UAI*, pages 366–374. AUAI Press, 2008.

S. Magliacane, T. Claassen, and J. Mooij. Ancestral causal inference. In *NIPS*, pages 4466–4474. Curran Associates, Inc., 2016.

B. Malone, M. Järvisalo, and P. Myllymäki. Impact of learning strategies on the quality of Bayesian networks: An empirical evaluation. In *UAI*, pages 562–571. AUAI Press, 2015.

B. M. Malone and C. Yuan. A depth-first branch and bound algorithm for learning optimal Bayesian networks. In *GKR 2013*, volume 8323 of *LNCS*, pages 111–122. Springer, 2014.

D. Margaritis and F. Bromberg. Efficient Markov network discovery using particle filters. *Comput. Intell.*, 25(4):367–394, 2009.

R. Neal. On deducing conditional independence from d-separation in causal graphs with feedback. *J. Artif. Intell. Res.*, 12:87–91, 2000.

J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.

J. Pearl and R. Dechter. Identifying independencies in causal graphs with feedback. In *UAI*, pages 420–426. Morgan Kaufmann, 1996.

K. Rantanen, A. Hyttinen, and M. Järvisalo. Learning chordal Markov networks via branch and bound. In *NIPS*, pages 1845–1855, 2017.

T. Richardson. A discovery algorithm for directed cyclic graphs. In *UAI*, pages 454–461. Morgan Kaufmann, 1996.

R. D. Shachter. Bayes-ball: Rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *UAI*, pages 480–487. Morgan Kaufmann, 1998.

P. Spirtes. Directed cyclic graphical representation of feedback models. In *UAI*, pages 491–498. Morgan Kaufmann, 1995.

P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. MIT Press, 2000.

M. Studený. Bayesian networks from the point of view of chain graphs. In *UAI*, pages 496–503. Morgan Kaufmann, 1998.

J. Suzuki. Learning bayesian belief networks based on the minimum description length principle: An efficient algorithm using the B & B technique. In *ICML*, pages 462–470. Morgan Kaufmann, 1996.

J. Suzuki and J. Kawahara. Branch and bound for regular Bayesian network structure learning. In *UAI*. AUAI Press, 2017.

J. Tian. A branch-and-bound algorithm for MDL learning Bayesian networks. In *UAI*, pages 580–588. Morgan Kaufmann, 2000.

S. Triantafillou and I. Tsamardinos. Constraint-based causal discovery from multiple interventions over overlapping variable sets. *J. Mach. Learn. Res.*, 16:2147–2205, 2015.

S. Triantafillou, I. Tsamardinos, and I. G. Tollis. Learning causal structure from overlapping variable sets. In *AISTATS*, pages 860–867. JMLR, 2010.

K. Tsirlis, V. Lagani, S. Triantafillou, and I. Tsamardinos. On scoring maximal ancestral graphs with the max-min hill climbing algorithm. In *KDD Workshop on Causal Discovery*, 2017.

P. van Beek and H. Hoffmann. Machine learning of Bayesian networks using constraint programming. In *CP*, volume 9255 of *LNCS*, pages 429–445. Springer, 2015.

C. Yuan and B. Malone. Learning optimal Bayesian networks: A shortest path perspective. *J. Artif. Intell. Res.*, 48:23–65, 2013.

Zhalama, J. Zhang, F. Eberhardt, and W. Mayer. SAT-based causal discovery under weaker assumptions. In *UAI*. AUAI Press, 2017.