Discovering Causal Graphs with Cycles and Latent Confounders: An Exact Branch-and-Bound Approach

Kari Rantanen, Antti Hyttinen, Matti Järvisalo[1]

Helsinki Institute for Information Technology HIIT,
Department of Computer Science,
University of Helsinki, Finland

Abstract

Understanding causal relationships is a central challenge in many research endeavours. Recent research has shown the importance of accounting for feedback (cycles) and latent confounding variables, as they are prominently present in many data analysis settings. However, allowing for cycles and latent confounders makes the structure learning task especially challenging. The constraint-based approach is able to learn causal graphs even over such general search spaces, but to obtain high accuracy, the conflicting (in)dependence information in sample data need to be resolved optimally. In this work, we develop a new practical algorithmic approach to solve this computationally challenging combinatorial optimization problem. While recent advances in exact algorithmic approaches for constraint-based causal discovery build upon off-the-shelf declarative optimization solvers, we propose a first specialized branch-and-bound style exact search algorithm. 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 and reasoning rules, and branching heuristics together with linear programming based bounding techniques, as well as directly incorporating different constraints on the search space, such as sparsity and acyclicity constraints. We empirically evaluate our implementation of the approach, showing that it outperforms current state of art in exact constraint-based causal discovery on real-world instances.

Keywords: Graphical models; structure learning; causal discovery; branch and bound; optimization.

Email addresses: kari.rantanen@helsinki.fi (Kari Rantanen), antti.hyttinen@helsinki.fi (Antti Hyttinen), matti.jarvisalo@helsinki.fi (Matti Järvisalo)

[1]Corresponding author

1. Introduction

Discovering causal relations from sample data when allowing for latent confounding variables and feedback (that is, 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 general search spaces (allowing latent confounders and/or cycles), the constraint-based causal discovery approach is still applicable (Spirtes et al., 2000; Pearl, 2000). Constraint-based learning algorithms combine (in)dependence constraints from statistical tests to find determined features of the underlying causal graph structure. However, most of such approaches, including the classical PC, CCD and FCI algorithms, scale up in terms of number of variables by selecting independence tests based on earlier test results (Spirtes et al., 2000; Richardson, 1996a). 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).

On the other hand, for restricted settings without latent confounders and cycles, that is, for Bayesian networks, exact score-based structure discovery algorithms have been developed (Yuan and Malone, 2013; Bartlett and Cussens, 2017; van Beek and Hoffmann, 2015). A central motivation in developing efficient exact algorithms is that they output a guaranteed optimal solution without making compromises or approximations in their computation. Such *provably globally optimal graphs* have been shown to exhibit better accuracy (Malone et al., 2015). However, much less progress has been made for exact discovery algorithms for more general search spaces that allow for latent confounders and cycles.

In the context of constraint-based discovery, it has been shown that better accuracy can be obtained when a predetermined, large set of tests are conducted before the actual search, and then, 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 search 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 (DAGs). Furthermore, the objective functions considered are computationally more complicated to evaluate. Thus improvements to (exact) algorithms for the more general search spaces in terms of running time performance and scalability without trading off accuracy is a major challenge.

In this work, we take on the challenge of improving the scalability of practical exact algorithms for the general search space of causal graph allowing for latent confounding variables and cycles. Recently, there has been noticeable interest in developing algorithmic solutions to this general problem setting and its variants (Triantafillou et al., 2010; Triantafilou et al., 2010; Hyttinen et al., 2013, 2014; Magliacane et al., 2016; Borboudakis and Tsamardinos, 2016; Zhalama et al., 2017; Hyttinen et al., 2017a). The first exact approach to the problem

we focus on here was proposed in (Hyttinen et al., 2014), based on declaratively encoding the underlying optimization task as answer set programming (ASP) and applying an ASP solver to obtain provably optimal solutions to the problem. This approach was further refined as a maximum satisfiability (MaxSAT) based approach in (Hyttinen et al., 2017b), where domain-specific techniques were integrated to the extent possible to a MaxSAT solver, relying on a MaxSAT solver to solve the search problem starting with a declarative encoding of the problem. This resulted in the *Dseptor* system which currently represents the state of the art in terms of running time performance for the problem at hand. All in all, this line of work has so far focused on using declarative solving techniques, relying in terms of efficiency on generic off-the-shelf declarative methods such as Boolean satisfiability (SAT) (Biere et al., 2009) solvers and their extensions to Boolean optimization. While declarative methods offer flexibility and remove implementation-level burden of developing optimized search algorithms for the underlying combinatorial optimization tasks, in this work we explore the alternative of developing domain-specific search algorithms instead of directly relying on declarative solver to perform the search.

In this paper we propose a first specialized branch-and-bound style exact search algorithm for optimal causal graphs, allowing the presence of 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 search space, such as sparsity and acyclicity constraints. In particular, we develop a branch-and-bound approach to directly search over the general search space, together with several different performance-improving search techniques. These include (i) a problem-specific branching heuristic, (ii) lower bounding techniques applicable during search based on problem-specific unsatisfiable cores and linear programming relaxations, (iii) optimized algorithms for evaluating the objective function of the problem—over exponentially many independence and dependence constraints—during search under partial solutions, and (iv) inference rules—with correctness proofs—for detecting which edges are irrelevant in terms of d-connectivity under a current partial solution. We provide an open-source implementation *bcause* of the approach, and empirically evaluate its performance on problem instances obtained from real-world datasets from several perspectives: (i) the marginal contribution of the different proposed search techniques, (ii) the impact of the scoring function used for obtaining constraint weights on the efficiency of the approach, and (iii) the efficiency of the approach with respect to current state of the art. In particular, we show that the proposed approach compares favourably with current state of the art in exact constraint-based causal discovery on real-world data sets with respect to running time performance.

This article considerably extends a preliminary version published at the PGM 2018 conference (Rantanen et al., 2018). In particular, in this article we describe more effective, earlier unpublished techniques for efficient evaluation of the objective function and formalize further inference rules which allow

for disregarding undecided edges under partial solutions during search, thereby further speeding up the overall search for an optimal causal graph. We have now implemented these new techniques in a new release version of the *bcause* system. Empirical results presented here have been obtained using this new version; compared to the version presented at PGM 2018, the additional techniques presented in this article have resulted in non-negligle running time improvements (obtaining up to 10x speed-up and 2x average speed-up) over the version of the system presented at PGM 2018. We have also considerably extended the empirical evaluation of the approach with earlier unpublished results: we present empirical data on the marginal contributions of the various search techniques implemented in *bcause* to the overall efficiency of the approach in practice, as well as a running time comparison with the earlier state-of-the-art *Dseptor* system (Hyttinen et al., 2017b). In addition to these new technical contributions, we have considerably extended the discussion and included various examples for improved readability and self-containment.

The rest of this article is organized as follows. We begin by detailing the necessary background on causal discovery, including causal graphs with latent variables and cycles, the combinatorial optimization task of finding optimal causal graph, and approaches for obtaining well-defined objective function coefficients in terms of weights on the independence and dependence constraints (Section 2). We then continue with detailing the proposed branch-and-bound approach to optimal causal graphs and several efficiency-improving search techniques for the approach (Section 3). We present results from an extensive empirical evaluation of the approach in Section 4). Before conclusions, we discuss the connections of our contributions to related work (Section 5).

2. Constraint-based Causal Discovery

In this section we give necessary background on causal graphs and the exact problem definition for the structure discovery task we consider in this work.

2.1. Causal Graphs

Causal structure can be represented by directed graphs where directed edges denote causal relations and nodes correspond to random variables for different measurements (Pearl, 2000; Spirtes et al., 2000). Although graphs are sometimes restricted to be acyclic, here we allow for directed cycles to be able to represent feedback (Spirtes, 1995; Richardson, 1996a,b).

In most analysis situations, we are not able to observe all relevant variables or all background factors. Fortunately, the use of bi-directed edges allow for a canonical representation of causal structures as a graph over the observed variables (Pearl, 2000; Spirtes et al., 2000). A bidirected edge $X \leftrightarrow Z$ represents a *latent confounder*, e.g. structure $X \leftarrow L \rightarrow Z$, where L is an unmeasured common cause of two observed variables X and Z. This prompts us to use the following graphs to represent causal structures.

Definition 1 (Causal graphs)**.** *A causal graph is a pair $G = (\mathcal{V}, E)$ with set of nodes $\mathcal{V}$, where the edge relation $E = E_{\rightarrow} \cup E_{\leftrightarrow}$ is composed of directed edges $E_{\rightarrow} \subseteq \mathcal{V} \times \mathcal{V}$ and (symmetric) bi-directed edges $E_{\leftrightarrow} \subseteq \{\{X, Y\} : X, Y \in \mathcal{V}\}$.*

The class of causal graphs is denoted by $\mathcal{G}$. Note that when the directed edges are not allowed to form cycles, causal graphs are semi-Markovian graphs (Pearl, 2000). Importantly, in both cases causal graphs are closed under marginalization.

The central reachability criterion for causal graphs is the following d-separation (Pearl, 2000). We follow here the definition of Studený (1998) which has been shown to be equivalent to Pearl's standard definition.

Definition 2 (d-separation)**.** *Two nodes X and Y in a causal graph $G = (\mathcal{V}, E)$ are d-connected given a conditioning set $C \subseteq \mathcal{V} \setminus \{X, Y\}$ if there is at least one d-connecting walk between them; otherwise they are d-separated. A* walk *is a sequence of 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 have an arrow head into the node. A walk is* d-connecting *given a conditioning set C if every collider on the walk is in C and no other nodes on the walk are in C.*

Example 1. *The causal structure in Figure 1 a) with unobserved L can be canonically represented by the causal graph in Figure 1 b). In the structure of Figure 1 a), X and W are d-connected given Y by $X \leftarrow L \rightarrow Z \rightarrow Y \leftarrow Z \leftarrow T \leftarrow W$. In the corresponding canonical representation in Figure 1 b), X and W are d-connected given Y by $X \leftrightarrow Z \rightarrow Y \leftarrow Z \leftarrow T \leftarrow W$. In the causal graph in Figure 1 b) nodes Y and Q are d-separated given W as all walks between violate the d-connection criterion at node X.*

Self-loops $X \rightarrow X$ do not affect d-connectivity of the graph: for any d-connecting walk through $X \rightarrow X$ there is a shorter walk that skips the arc $X \rightarrow X$. Thus, self-loops are inherently unidentifiable here; they are unidentifiable also in other settings (Lacerda et al., 2008; Hyttinen et al., 2012). Without loss of generality, we do not consider self-loops through the rest of this article. Similarly, without loss of generality we do not consider arcs $X \leftrightarrow X$. We emphasize that any $X \rightarrow X$ or $X \leftrightarrow X$ may be present in the true structure regardless of the result of the algorithmic approach developed in this article.

2.2. Statistical Dependence & Reachability in Graphs

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 on cyclic causal graphs applies under the following assumptions. The parametric models to cyclic graphs are non-recursive structural equation models (SEMs) (Wright, 1934; Bollen, 1989; Richardson, 1996b). 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,

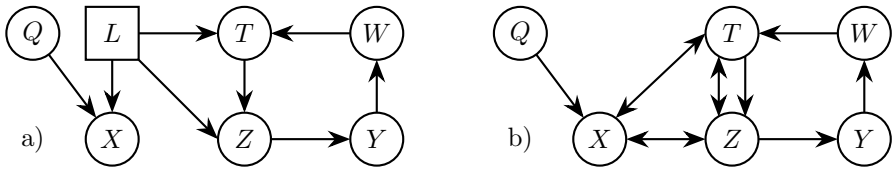

Figure 1: Example graphs: a) a causal graph with an unobserved node L, b) the canonical representation of a) using bidirected edges.

1995). The same result applies for discrete random variables, when the structural equations to every ancestral subset (a set of nodes and their ancestors) has a unique solution (Forré and Mooij (2017): Theorem 3.8.12 on page 112, see also Pearl and Dechter (1996), Neal (2000)).

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 conditionally dependent given a set of variables C if and only if they are d-connected given C in the generating causal structure G. In the rest of the article we use $X \perp\!\!\!\perp Y|C$ ($X \not\perp\!\!\!\perp Y|C$) to denote statistical independence (dependence) and d-separation (d-connection).

Example 2. *Given enough samples from a causal model with the 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.*

2.3. Problem Definition

In constraint-based causal discovery, the aim is to find an equivalence 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 given set of variables $\mathcal{V}$, and a non-negative weight $w(k)$ for each $k \in K$.

Task: Find a causal graph $G^* = (\mathcal{V}, E^*)$ such that

$$G^* \in \operatorname{argmin}_{G \in \mathcal{G}} \sum_{k \in K \,:\, G \not\models k} w(k). \tag{1}$$

In words, 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. Apart from this constraint satisfaction perspective, Section 2.5 gives a probabilistic motivation for this objective function.

Example 3. *Let the nodes be $\mathcal{V} = \{X, Y, Z\}$ and let the (in)dependence constraints K be as follows (weights in parenthesis):*

$X \perp\!\!\!\perp Y \mid Z$ *(1098)*	$Y \perp\!\!\!\perp Z \mid X$ *(97)*	$X \not\perp\!\!\!\perp Z$ *(106837)*
$X \not\perp\!\!\!\perp Z \mid Y$ *(101804)*	$Y \not\perp\!\!\!\perp Z$ *(4935)*	$X \not\perp\!\!\!\perp Y$ *(3935)*

This includes all relations testable in passively observed data over three variables.

As the example shows we include in K only one constraint for nodes $\{X, Y\}$ and set $C \subseteq \mathcal{V} \setminus \{X, Y\}$, either in the form of an independence $X \perp\!\!\!\perp Y|C$ or a dependence $X \not\perp\!\!\!\perp Y|C$. Several constraints for the same $\{X, Y\}$ and C can be compressed to a single constraint just by summing up the weights appropriately.

The score function trivially satisfies *score equivalence* (Heckerman et al., 1995): all Markov equivalent structures imply the same d-separations and therefore obtain the exact same score regardless of the weight is used (for a fixed set constraints K). Thus, an optimal causal graph G^* is a representative of the (Markov) equivalence class closest to the input constraints.

Solving this problem exactly has the following consistency result (Hyttinen et al., 2013, 2014). Under the assumptions discussed in Section 2.2 we have that statistical independence is equivalent to d-separation. When the weights are obtained by a test that consistently detects statistical dependence, we have that in the infinite sample limit, K includes independence and dependence relations that correspond respectively to d-separation and d-connection relations in the true graph. Consequently, the optimal solution will be in the equivalence class of the true graph and satisfy all constraints in K.

Completeness depends on what set K is used (Hyttinen et al., 2013). If not all testable relations are in K, there may be information in the additional relations in the data that allow for further identification of structural features. In this article we include in K all $\binom{n}{2}2^{n-2}$ relations testable in passively observed data of n variables. Therefore a structural feature determined by relations testable in the data will be uniquely determined in the equivalence class of the top scoring causal graph. The properties of the equivalence class can be studied for example with the SAT-based procedure of Hyttinen et al. (2013) or in the acyclic case by FCI (Spirtes et al., 2000).

2.4. Weights for Independence Constraints

The algorithmic approach developed in this article is agnostic in terms of how weights are obtained. One way to obtain weights is through Bayesian model selection (Cooper, 1997; Steck and Jaakkola, 2002; Abellán et al., 2006; Margaritis and Bromberg, 2009; Hyttinen et al., 2014). For each independence statement $X \perp\!\!\!\perp Y|C$, consider two models

$$\begin{aligned} M_{\perp\!\!\!\perp} &: P(X, Y|C) = P(X|C)P(Y|C) \\ M_{\not\perp\!\!\!\perp} &: P(X, Y|C) = P(X|C)P(Y|X, C) \end{aligned}$$

where the first postulates independence, and the second postulates dependence. Given data D on X, Y, C and a prior probability of independence $P(M_{\perp\!\!\!\perp}) = \alpha$

the probability associated with $k = X \perp\!\!\!\perp Y|C$ simplifies to

$$P(k|D) = \frac{P(Y|C)\alpha}{P(Y|C)\alpha + P(Y|X,C)(1-\alpha)}.$$

The marginal likelihoods $P(Y|C)$ and $P(Y|X,C)$ correspond directly the local scores in the score-based Bayesian network structure learning framework, which can be evaluated in closed form for categorical variables when using a Dirichlet prior (Buntine, 1991; Cooper and Herskovits, 1992) and for continuous variables with linear relations and Gaussian disturbances using an inverse Wishart Gaussian prior (Geiger and Heckerman, 2002). Note that since both scores are score-equivalent, the same probabilities are obtained if $M_{\not\perp\!\!\!\perp}$ uses factorization $P(X,Y|C) = P(X|Y,C)P(Y|C)$ instead.

Since we optimize the sum of violated constraints, for nodes X, Y and set C we include k (independence or dependence relation) that obtained the higher probability with weight obtained by the following log transformation:

$$w(k) = \log P(k|D) - \log P(\neg k|D) \tag{2}$$

There are also several alternative ways of obtaining weights that can be directly used by our procedure. Jabbari et al. (2017) use similar Bayesian model selection, but dependence is modeled by $P(Z|C)$ where Z is a random variable whose values are a Cartesian product of the values for X and Y. Natori et al. (2017) study the use of different priors. Also BIC approximations can be utilized (Hyttinen et al., 2017a). The approach of Claassen and Heskes (2012) obtains probabilities for d-separation relations by Bayesian model averaging over graphs. Triantafilou et al. (2010); Magliacane et al. (2016) employ frequentist statistical hypothesis testing to obtain similar reliability weights.

2.5. Motivation for the Objective Function

Apart from a constraint satisfaction perspective, the objective function in Equation 1 can be given a probabilistic motivation (Hyttinen et al. (2014): Appendix B, Jabbari et al. (2017): Section 4). The posterior probability of a graph G given data D can be written as

$$P(G|D) \quad = \quad \sum_{K_i \in \mathcal{K}} P(G|K_i, D)P(K_i|D),$$

where $\mathcal{K}$ includes all sets of (in)dependence constraints that can be obtained from the data.

The standard assumption underlying constraint-based causal discovery is that the (in)dependence constraints exhaust all information on the causal graph in the data, $G \perp\!\!\!\perp D|K_i$ (Jabbari et al., 2017; Hyttinen et al., 2014):

$$P(G|D) \quad = \quad \sum_{K_i} P(G|K_i)P(K_i|D).$$

Another standardly made assumption is that constraints are distributed independently given the data D (Claassen and Heskes, 2012; Hyttinen et al., 2014; Triantafilou et al., 2010; Magliacane et al., 2016; Jabbari et al., 2017):

$$P(G|D) \quad = \quad \sum_{K_i} P(G|K_i) \prod_{k \in K_i} P(k|D).$$

Note that this is different from only assuming mutual independence of constraints *unconditional on the data*.[2]

The term $P(G|K_i)$ is non-zero only for the constraints $K_i = K_G$ implied by G. Since an independence constraint in K_G may correspond to a dependence in K and vice versa, we further have

$$P(G|D) \quad = \quad \prod_{k \in K_G} P(k|D) = \prod_{k \in K \,:\, G \models k} P(k|D) \prod_{k \in K \,:\, G \not\models k} P(\neg k|D).$$

For finding the optimal G, we can take the logarithm to obtain

$$\log P(G|D) \quad = \quad \sum_{k \in K \,:\, G \models k} \log P(k|D) + \sum_{k \in K \,:\, G \not\models k} \log P(\neg k|D),$$

and subtract term $\sum_{k \in K} \log P(k|D)$ that is constant with respect to G, obtaining

$$\log P(G|D) - \sum_{k \in K} \log P(k|D) = \sum_{k \in K \,:\, G \not\models k} [\log P(\neg k|D) - \log P(k|D)] = -\sum_{k \in K \,:\, G \not\models k} w(k),$$

where $w(k)$ is defined as in Equation 2. Thus, under these modeling assumptions, maximizing posterior probability of a graph given the data $P(G|D)$ is equivalent to minimizing the objective in Equation 1.

3. Branch and Bound for Constraint-based 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 an efficient method for determining the satisfied/violated (in)dependence constraints in each search tree branch (Section 3.2), an effective domain-specific branching heuristic (Section 3.4), and how to obtain tight bounds during search using linear programming relaxations (Section 3.5). Furthermore, we describe how structural restrictions on the search space, such as enforcing acyclicity and degree restrictions (Section 3.6), can be integrated. Before detailing these techniques, we start with an overview of the core branch-and-bound routine.

[2] Jabbari et al. (2017) use a sampling-based approach to account for dependencies among input constraint in an inexact approach. In their simulations the solutions closely corresponded to the solutions assuming independent constraints given the data.

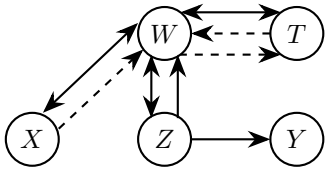

Figure 2: A partial solution; solid edges have been decided present, dashed edges remain undecided, others are decided absent.

3.1. Overview

The overall structure of the branch-and-bound search is presented as Algorithm 1. A The algorithm performs a complete depth-first search over the causal graphs within the well-known general algorithmic framework of branch and bound, extending partial solutions towards fully defined causal graphs, and using bounding techniques for pruning out partial solutions which can be determined not to improve the current best solution.

In this context, a *partial solution* G is a graph in which each edge is either decided *absent*, decided *present* or *undecided*.

Example 4. *Consider the illustration of a partial solution in Figure 2. The solid edges and absent edges have been decided to be present and abstract, respectively. The dashed edges represent undecided edges in the partial solution, meaning that (in case the search branch represented by the partial solution is not pruned out before this) the search will subsequently traverse over the subsearch space spanned by the undecided edges.*

At each search tree node, on Line 2 we compute a lower bound for the weight of the current 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 backtrack. 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 edges exist, that is, $e^* = null$, 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 heuristically (see Section 3.4 for details). 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

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
```

one where (b) the edge is decided absent. The order in which we visit these branches is determined heuristically, see Section 3.4 for details. At the end of the search, G^* is guaranteed to be a solution with globally optimal cost.

For computing a simple initial upper bound solution G^*, we first initialize it as an empty graph, then traverse the 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 the graph as long as this locally improves the weight of G^*. Any edge addition which would make G^* violate possible search space restrictions (Section 3.6) is omitted.[3]

We will now provide an example of how our branch-and-bound search would behave with a simple 3-variable instance.

Example 5. *Let the nodes be $\mathcal{V} = \{X, Y, Z\}$ and let the (in)dependence constraints K be as follows (weights in parenthesis):*

$X \perp\!\!\!\perp Y \mid Z$ (1098)	$Y \perp\!\!\!\perp Z \mid X$ (97)	$X \not\perp\!\!\!\perp Z$ (106837)
$X \not\perp\!\!\!\perp Z \mid Y$ (101804)	$Y \not\perp\!\!\!\perp Z$ (4935)	$X \not\perp\!\!\!\perp Y$ (3935)

Before entering the search itself, we construct the initial upper bound solution. For this purpose we traverse the four dependence constraints in descending weight order and add the edges $X \to Z$, $Y \to Z$ and $X \to Y$ to an empty graph. Each edge addition locally improves the solution's weight and there are no more variable pairs (with dependence constraints) and as such the resulting graph (shown in Figure 3 (0)) serves as our initial upper bound solution with weight 1195.

We are now ready to enter the branch and bound. The steps 1-5 of the search are illustrated in Figure 3. We start off with a partial solution where all the edges are undecided (Step 1). Assume that we use a branching strategy where we branch first by deciding edges absent between variables that seem most likely to be independent. Concretely, we first decide the edges $X \leftrightarrow Y$, $X \to Y$ and $X \leftarrow Y$ to be absent (Step 2), followed by $Y \leftrightarrow Z$, $Y \to Z$ and $Y \leftarrow Z$ (Step 3). However, after deciding $Y \leftarrow Z$ absent, we obtain a lower bound of 8870 for the partial solution, which is larger than our incumbent upper bound of 1195. Hence we backtrack, deciding $Y \leftarrow Z$ to be present instead (Step 4).

We continue the search by deciding $X \leftrightarrow Z$, $X \to Z$ and $X \leftarrow Z$ to be present (Step 5), as our branching heuristic recognizes that there are no independence constraints between X and Z. Now there are no more edge decisions to be made, and so we evaluate the solution at hand. It violates only the constraint $[Y \perp\!\!\!\perp Z \mid X]$ and thus has weight 97. This is better than our previous solution with weight of 1195, and hence we update our incumbent upper bound to this new one.

Next we backtrack in the search. For each decision we made in the search tree (except for the presence of $Y \leftarrow Z$), we also have a branch with the opposite decision (i.e., deciding a present edge to be absent, or deciding an absent edge to

[3] In practice, however, we have observed empirically that the thereby obtained initial upper bound tends to have only a negligible impact on overall runtime performance of the approach.

be present). However, immediately after choosing any of these alternatives, we obtain a lower bound which closes the corresponding branch. Hence we backtrack all the way to the root node of the search tree, closing all the branches on the way, thus determining that the found graph with weight 97 is indeed the optimal solution.

3.2. Efficient Evaluation of the Objective Function

Given that there are superpolynomially many (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 extending 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

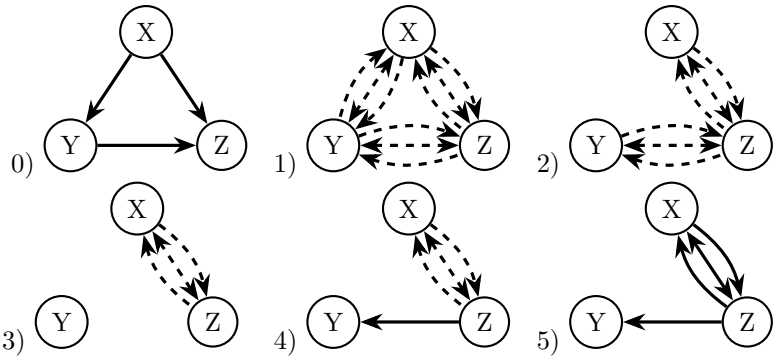

Figure 3: The phases of the example. (0): The initial upper bound solution. (1-5): Gradual construction of a solution in the search.

Algorithm 2 Efficient update of constraint states after an edge decision.

```
1: function CheckConstraints(partial solution G, nodes X, Y, edge e )
2:     if e is present in G then
3:         G' ← minc(G)
4:     else
5:         G' ← maxc(G) with e
6:     if e does not affect the d-connectivity of X and Y in G' then return
7:     C+ ← Unavoidable colliders on d-connections between X and Y in G'.
8:     C− ← Unavoidable non-colliders on d-connections between X and Y.
9:     Check constraints of the form [X ⫫ Y | S] (and [X ⫫̸ Y | S])
10:        such that C+ ⊆ S and S ∩ C− = ∅.
```

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* $\text{maxc}(G)$ of a partial solution has all undecided edges marked present (e.g. Figure 2 *with* the dashed edges), a *minimal completion* $\text{minc}(G)$ has all undecided edges absent (Figure 2 *without* the dashed edges). An independence constraint is satisfied if the corresponding d-separation holds in $\text{maxc}(G)$, and violated if the corresponding d-separation does not hold in $\text{minc}(G)$ (Hyttinen et al., 2013). A dependence constraint is satisfied if the corresponding d-connection holds in $\text{minc}(G)$, and violated if the corresponding d-connection does not hold in $\text{maxc}(G)$. All other constraints are undetermined. In the beginning of the search, when no edges are decided, the states of all constraints are undetermined.

Example 6. *Consider the partial solution in Figure 2. The constraint* $[X \not\perp\!\!\!\perp Z \mid W]$ *is satisfied, since no matter how one decides the undecided edges, the path* $X \to W \leftarrow Z$ *will always exist in the resulting graph. On the other hand, the constraint* $[W \not\perp\!\!\!\perp Y \mid Z]$ *is violated, since no matter how one decides the undecided edges, all the paths between* W *and* Y *in the resulting graph will contain* Z *as a non-collider. Moreover, the constraint* $[Z \perp\!\!\!\perp T]$ *is undetermined since it can be either satisfied or violated depending on whether the undecided edge* $W \to T$ *is decided present or absent.*

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 stronger, what we call core-based lower bounds (as detailed in Section 3.5) stay up to date. Note that the choice of how regularly we update constraint states from undetermined to satisfied/violated 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 acyclicity or an edge degree limit (see Section 3.6) 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. For determining whether a d-connection exists between two nodes X and Y given some conditioning set C, we use a straightforward algorithm that simply checks whether there is a path between X and Y where all the colliders are in C and no other nodes are in C.

An efficient way to update constraint states for a given node pair after an edge decision is presented as 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 given some C in the minimal completion G' (Line 6). If not, the constraints need not be updated, because deciding an edge present does not remove existing d-connections, and hence the constraint states

between the node pair remain unchanged. Otherwise, if new d-connections may have been formed, we identify a set of unavoidable colliders C^+ and non-colliders C^- between all d-connecting walks between X to Y in G' (Line 7 and 8). We can then omit checking any constraint states for X and Y where conditioning set does not contain all the colliders C^+ or contains some non-colliders C^-. This is because, by the definition of d-separation, d-connections that contradict these collider/non-collider requirements cannot have been formed in the completion. 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, because this information is merely used to speed up the constraint evaluation, and it does not affect the end-result (i.e., the determined states) of the evaluation. Hence we need to make a trade-off 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 straightforward method described in Algorithm 3 for gathering only some (i.e., in general not all) of the unavoidable colliders/noncolliders when traversing from node X to Y in the (minimal/maximal) completion G' of partial solution G. Here the node A is the starting point and V is the set of nodes that have already been visited. We execute this procedure starting from both X and Y; i.e., FindUnavoidables$(G', X, \emptyset, X, Y)$ and FindUnavoidables$(G', Y, \emptyset, X, Y)$. For a graph with n nodes, each of these procedure calls have a $\mathcal{O}(n^2)$ time complexity and a $\mathcal{O}(n)$ space complexity.

Deciding an edge e absent is essentially analogous to the case where the edge is decided present. However, as mentioned in Algorithm 2, in this case we have to make sure that e still exists in the maximal completion G'. This is because

Algorithm 3 A simple method for finding unavoidable colliders and non-colliders between nodes X and Y in a completion G' starting from node $A \in \{X, Y\}$.

1: **function** FindUnavoidables(graph G', nodes A, X, Y, set V of visited nodes)
2: $\quad\mathcal{N} \leftarrow$ neighbours of A in G'
3: $\quad$**if** $A \notin \{X, Y\}$ **then**
4: $\quad\quad$**if** for all $N \in \mathcal{N}$ the edge $A \to N$ is not present in G' **then**
5: $\quad\quad\quad C^+ \leftarrow C^+ \cup \{A\}$
6: $\quad\quad$**else if** for all $N \in \mathcal{N} \setminus V$ neither $A \leftarrow N$ nor $A \leftrightarrow N$ is present **then**
7: $\quad\quad\quad C^- \leftarrow C^- \cup \{A\}$
8: $\quad$**if** $|\mathcal{N} \setminus V| \neq 1$ or $\mathcal{N} \setminus V \subseteq \{X, Y\}$ **then**
9: $\quad\quad$**return**
10: $\quad$Let $B \in \mathcal{N} \setminus V$
11: $\quad$**if** neither $A \to B$ or $A \leftrightarrow B$ is present in G' **then**
12: $\quad\quad C^- \leftarrow C^- \cup \{B\}$
13: $\quad$Call FindUnavoidables$(G', B, V \cup \{A\}, X, Y)$

here we are interested in checking which d-connections get removed from the completion due to the edge decision, and hence we want use the unavoidable nodes C^+ and C^- that existed *before* the edge was removed from the completion. That is, the edge can only have removed d-connections which satisfy these collider/non-collider requirements.

Example 7. *For the partial solution in Figure 2, 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. This tells us that we do not need to check the existence of d-connections between X and Y that have Z in the conditioning set. That is, we must have $X \perp\!\!\!\perp Y \mid Z$; $X \perp\!\!\!\perp Y \mid Z, W$; $X \perp\!\!\!\perp Y \mid Z, T$ and $X \perp\!\!\!\perp Y \mid Z, W, T$.*

3.3. Rules for Inferring Irrelevant Edges

In this section we provide a way to reduce the amount of time the search has to spend at each search tree node. Recall that at each step of the search, we decide an edge e to be either present or absent in the current partial solution G, and then use the completions $\text{minc}(G)$ and $\text{maxc}(G)$ to determine whether states of the (in)dependence constraints changed. Here we provide simple rules for detecting when an edge decision by itself cannot affect the states of (certain) constraints. We then use this information to avoid performing a considerable amount of unnecessary d-separation checks which would create unwanted overhead to the search.

The rules are based on so called *inducing paths* (Verma and Pearl, 1990; Spirtes et al., 2000). To allow for the development of theory relevant to the current setting, we define inducing paths in this paper as follows.

Definition 3. *Let X and Y be nodes in a causal graph $G = (\mathcal{V}, E)$. If there exist distinct $V_1, \ldots, V_k \in \mathcal{V}$ where $V_1 = X$ and $V_k = Y$ such that*

- *$V_1 \to V_2$ is present, and*
- *both $V_{i-1} \to V_i$ and $V_{i-1} \leftrightarrow V_i$ is present for each $i > 2$,*

then we denote $X \rightsquigarrow Y$.

The following lemma will be useful for arguing about the correctness of the rules (Lemma 3 of Verma and Pearl (1990)).

Lemma 1. *Vertices A and B are d-connected in graph $G = (\mathcal{V}, E)$ given any conditioning set $C \subseteq \mathcal{V} \setminus \{A, B\}$ if $A \rightsquigarrow B$ is present in G.*

Proof. Let $A \rightsquigarrow B$. By the definition, there exists $V_1, \ldots, V_k \in \mathcal{V}$ for some $k \geq 2$ such that $V_1 = A, V_k = B$ where $V_1 \to V_2$ is present, and both $V_{i-1} \to V_i$ and $V_{i-1} \leftrightarrow V_i$ is present for each $i > 2$. When $k = 2$, the lemma's claim is trivial. When $k = 3$, we have the edges $A \to V_2 \to B$ and $V_2 \leftrightarrow B$ in G, so A and B are clearly connected given C regardless whether $V_2 \in C$.

Assume that the claim holds for some $k \geq 3$. We show that it holds for $k + 1$. We have $A \rightsquigarrow V_k \to B$ and $V_k \leftrightarrow B$ in G, so A and V_k are d-connected

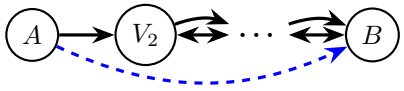

Figure 4: Illustration of of the edge irrelevancy rule 1. The dashed edge corresponds to the edge whose impact to the d-connectivity is to be checked. Note that the rule would trigger even if the graph would contain arbitrary additional nodes and edges.

given C. Now, if $V_k \in C$, choose d-connecting path $A \rightsquigarrow V_k \leftrightarrow B$, otherwise $A \rightsquigarrow V_k \rightarrow B$. Hence A and B are d-connected in G given C. □

We will now formally define four rules which allow for inferring that, given that particular edges are present/absent in the current partial solutions, the presence or absence of a currently undecided edge is irrelevant in terms of d-connectivity.

The first rule captures situations where adding an edge would form a 'shortcut' between two nodes that are already connected by an inducing path. Particularly, if the added edge has the same direction as the inducing path, then the edge does not increase d-connectivity in the graph since one could always use the inducing path instead of the shortcut.

Rule 1. *Let A and B be nodes in a causal graph such that $A \rightsquigarrow B$ is present. Then adding the edge $A \rightarrow B$ does not affect the d-connectivity of the graph.*

For an illustration of Rule 1, see Figure 4. In the figure, the dashed edge corresponds to the edge whose impact to the d-connectivity is to be checked. Note that the rule would trigger even if the graph would contain arbitrary additional nodes and edges.

Proof. (Correctness of Rule 1) Assume that $A \rightsquigarrow B$ exists in G. Consider the path $A \rightarrow B$ and all the paths via $A \rightsquigarrow B$. In both cases, for each conditioning set $C \subseteq \mathcal{V} \setminus \{A, B\}$, there exists a path which d-connects A and B in G (by Lemma 1), and in all the paths (1) an edge is pointing outwards from A, and (2) an edge is pointing inwards to B. Hence the edge $A \rightsquigarrow B$ does not affect the d-connectivity in G. □

The second rule captures situations where the added 'shortcut' edge between two nodes is bi-directional. In this case the edge and the inducing path point differently at one of the nodes and similarly to the other node. Intuitively, the side of the structure in which the edge and the inducing path agree behaves similarly to the first rule. The side in which the edge and the inducing path disagree requires us to make sure that the shortcut does not create a new potential collider.

Rule 2. *Let A and B be nodes in a causal graph such that $A \rightsquigarrow B$ is present. Adding the edge $A \leftrightarrow B$ does not affect the d-connectivity of nodes X and Y if*

1. *$A = X$ or $A = Y$, or*

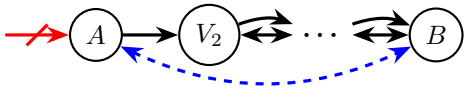

Figure 5: Illustration of of the edge irrelevancy rule 2. The dashed edge corresponds to the edge whose impact to the d-connectivity is to be checked. The crossed-out edge indicates that the target node cannot have incoming edges apart from the dashed one. Note that, apart from the restrictions denoted by the crossed-out edge, the rule would trigger even if the graph would contain arbitrary additional nodes and edges.

2. *A has no incoming edges besides from B.*

For an illustration of Rule 2, see Figure 5. The crossed-out edge indicates that the target node cannot have incoming edges apart from the ones in the figure that are not crossed out. Note that, apart from the restrictions denoted by the crossed-out edge, the rule would trigger even if the graph would contain arbitrary additional nodes and edges.

Proof. (Correctness of Rule 2) Assume that $A \rightsquigarrow B$ is present in G. Consider the path $A \leftrightarrow B$ and all the paths via $A \rightsquigarrow B$. In both cases, for each conditioning set $C \subseteq \mathcal{V} \setminus \{A, B\}$, there exists a path which d-connects A and B in G (by Lemma 1), and in all the paths an edge is pointing inwards to B in a path. The key difference is that in the path $A \leftrightarrow B$ there is an edge pointing to A, whereas on all paths via $A \rightsquigarrow B$ there is no edge pointing to A.

Assume first that $A = X$ or $A = Y$. Now, A cannot be a collider or a non-collider on d-connecting paths via X and Y, and so it does not matter whether or not an edge points to A. Hence $A \leftrightarrow B$ does not affect the d-connectivity of X and Y.

Assume then that A has no incoming edges besides from B (and $A \neq X, B \neq Y$). Now, A is a non-collider on all paths from X and Y which go through the path $A \leftrightarrow B$ or go through any path of $A \rightsquigarrow B$. Hence $A \leftrightarrow B$ does not affect the d-connectivity of X and Y. □

The third rule captures situations where the added 'shortcut' edge between two nodes points to the opposite direction wrt. the inducing path. Intuitively, since the edge and the inducing path disagree on both sides of the structure, this corresponds to two instances of the situation from the second rule where a node is pointed differently by the edge and the path.

Rule 3. *Let A and B be nodes in causal graph such that $A \rightsquigarrow B$ is present. Adding the edge $B \leftarrow A$ does not affect the d-connectivity of nodes X and Y if both of the following hold:*

1. *$A = X$ or $A = Y$ or A has no incoming edges besides from B, and*

2. *$B = X$ or $A = Y$ or B has no incoming edges besides from A.*

For an illustration of Rule 3, see Figure 6. Again, the crossed-out edges indicate that the target node cannot have incoming edges apart from the dashed one;

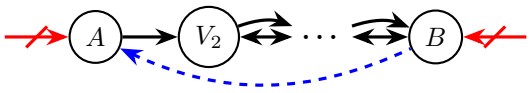

Figure 6: Illustration of of the edge irrelevancy rule 3. The dashed edge corresponds to the edge whose impact to the d-connectivity is to be checked. The crossed-out edges indicate that the target node cannot have incoming edges apart from ones in the figure that are not crossed out. Note that, apart from the restrictions denoted by the crossed-out edges, the rule would trigger even if the graph would contain arbitrary additional nodes and edges.

apart from the restrictions denoted by the crossed-out edge, the rule would trigger even if the graph would contain arbitrary additional nodes and edges.

Proof. (Correctness of Rule 3) Assume that $A \rightsquigarrow B$ is present in G. Consider the path $A \leftarrow B$ and all the paths via $A \rightsquigarrow B$. In both cases, for each conditioning set $C \subseteq \mathcal{V} \setminus \{A, B\}$, there exists a path which d-connects A and B in G (by Lemma 1). The key differences are that (1) in $A \leftarrow B$ there is an edge pointing to A whereas in all paths via $A \rightsquigarrow B$ there is not, and (2) in all paths via $A \rightsquigarrow B$ there is an edge pointing to B whereas in $A \leftarrow B$ there is not.

By applying similar reasoning as in the proof of Theorem 2, we have that if $A = X$ or $A = Y$ or A has no incoming edges besides from B, then on all paths from X to Y which go through the path $A \leftarrow B$ or go through any path via $A \rightsquigarrow B$, the d-connectivity of X and Y is not affected by whether an edge points to A. The same holds symmetrically in the case where $B = X$ or $B = Y$ or B has no incoming edges from A. Therefore, when both conditions of the rule hold, then the d-connectivity of X and Y is not affected by whether there is an edge pointing to A or B, and hence $A \leftarrow B$ does not affect the d-connectivity of X and Y in G. □

Similarly as Rule 1, Rules 2 and 3 can be used globally without concerning what the variables X and Y are. That is, if merely the 'has no incoming edges' conditions hold, we can omit checking the (in)dependence constraints for all variables.

Finally, the following fourth rule states that checking constraint states with respect to a maximal completion is unnecessary unless the completion is sparse enough.

Rule 4. *Removing an edge e from a causal graph does not affect the d-connectivity of X and Y if (1) $X \rightsquigarrow Y$, (2) $Y \rightsquigarrow X$ or (3) $X \leftrightarrow Y$ would still hold in the graph after removing e.*

Proof. (Correctness of Rule 4) When any of the three conditions hold, X and Y are d-connected given any conditioning set, regardless of e. Note that edges $X \rightarrow Y$ and $Y \rightarrow X$ are special cases of the first two conditions. □

All four edge relevancy rules introduced in this section are straightforward to check in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space, where n is the number of nodes in the graph. This is because the computationally most demanding part is just to verify whether an inducing path exists between two nodes in the graph.

3.4. 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

$$K^+(X) = \bigcup_Y K^+(X,Y) \quad \text{and} \quad K^-(X) = \bigcup_Y K^-(X,Y)$$

to denote the undetermined dependence and independence constraints involving node X, respectively. 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 in order 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.

For a graph with n nodes, it takes $\mathcal{O}(n^2)$ time and space to select the next node pair to branch with, assuming that the compared values ($w^+(\cdot)$, $\max_{k \in K^+(\cdot)} w(k)$ etc.) have been precomputed into cache. Indeed, we gather this information while evaluating the constraint states (Section 3.2), resulting in no additional complexity.

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 if and only if

$$[X \perp\!\!\!\perp Y \mid Z] \in K \quad \text{for any} \quad Z \subseteq \mathcal{V} \setminus \{X, Y\}.$$

(i) $\{X \perp\!\!\!\perp Z|S;\ X \perp\!\!\!\perp Y|S;\ X \not\perp\!\!\!\perp Z|Y,S\}$

(ii) $\{X \not\perp\!\!\!\perp Z|S;\ Y \not\perp\!\!\!\perp Z|S;\ X \perp\!\!\!\perp Y|S;\ X \perp\!\!\!\perp Y|Z,S\}$

(iii) $\{X \not\perp\!\!\!\perp Z|Y,S;\ Y \not\perp\!\!\!\perp Z|X,S;\ X \perp\!\!\!\perp Y|S;\ X \perp\!\!\!\perp Y|Z,S\}$

(iv) $\{Y \not\perp\!\!\!\perp Z|S;\ X \not\perp\!\!\!\perp Z|S;\ Z \perp\!\!\!\perp W|X,Y,S;\ X \perp\!\!\!\perp Y|Z,S;\ X \perp\!\!\!\perp Y|W,S\}$

(v) $\{Y \not\perp\!\!\!\perp Z|S;\ X \not\perp\!\!\!\perp Z|S;\ Z \perp\!\!\!\perp W|Y,S;\ X \perp\!\!\!\perp Y|Z,S;\ X \perp\!\!\!\perp Y|W,S\}$

(vi) $\{X \not\perp\!\!\!\perp Y|Z,S;\ Y \not\perp\!\!\!\perp Z|X,W,S;\ W \not\perp\!\!\!\perp Y|Z,S;$
$W \perp\!\!\!\perp X|Y,Z,S;\ X \perp\!\!\!\perp Z|W,S\}$

(vii) $\{X \not\perp\!\!\!\perp Y|Z,S;\ Y \not\perp\!\!\!\perp Z|X,W,S;\ W \not\perp\!\!\!\perp Y|S;\ W \perp\!\!\!\perp X|Y,S;\ X \perp\!\!\!\perp Z|W,S\}$

Figure 7: Small minimal unsatisfiable core patterns used for computing lower bounds via linear programming.

3.5. Computing Tight Lower Bounds by Linear Programming

We now describe how we compute strong lower bounds using core patterns (Hyttinen et al., 2017b). An *unsatisfiable core* is a set of (in)dependence constraints that cannot be simultaneously satisfied by any graph in $\mathcal{G}$. Some example cores are marked by rectangles in Figure 8. We use the seven core patterns from (Hyttinen et al., 2017b), shown in Figure 7, 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[4] where the unsatisfiable 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 a constraint from each core. To obtain the bounds in practice, similarly as in Hyttinen et al. (2017b), we solve linear relaxations of the following standard integer programming formulation of these hitting set problems using a linear programming (LP) solver.

Concretely, the objective of the integer program (IP) formulation of the minimum-cost hitting set problem is

$$\min \sum_{k \in K} w(k) \cdot x_k,$$

where each binary variable $x_k \in \{0, 1\}$ indicates whether the (in)dependence constraint $k \in K$ is included in the hitting set. In the linear relaxation, we have $x_k \in [0, 1]$.

[4]Given a collection of sets over a set of weighted elements, a minimum-cost hitting set is a subset of the elements such that (i) the hitting set contains at least one element form each of the sets in the collection, and (ii) the sum of the weights of the elements in the hitting set in smallest among all hitting sets of the collection.

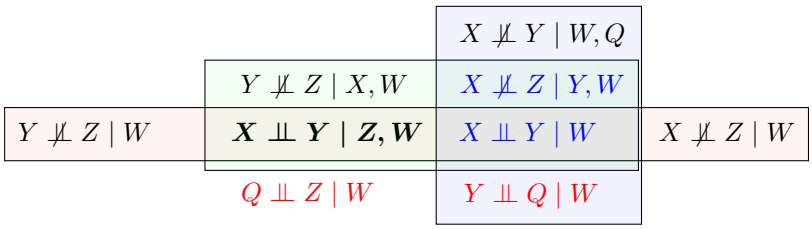

Figure 8: Example of core-based lower bounding.

The unsatisfiable cores form the linear constraints of the IP: for each unsatisfiable core over a set of (in)dependence constraints $k_1, k_2, \ldots, k_m \in K$, we include the corresponding linear constraint

$$x_{k_1} + x_{k_2} + \ldots + x_{k_m} \geq 1$$

to the program, enforcing that at least one of the (in)dependence constraints needs to be included in a hitting set.

For obtaining bounds via the linear relaxation of the minimum-cost hitting set IP at a search node during the branch-and-bound search, we simplify the linear relaxation by enforcing 0/1 values on those LP variables corresponding to known (in)dependence constraint states (recall Section 3.2) under the current partial solution. In particular, if a constraint $k \in K$ is known to be satisfied (violated, respectively) under the current partial solution, we enforce $x_k = 0$ ($x_k = 1$, respectively) in the linear relaxation, stating that we are not allowed to (we must, respectively) choose k into the hitting set. This way the core-based lower bounds are updated to match the current search tree branch.

Example 8. *Suppose we have the cores in Figure 8 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 the rectangles must be chosen. The violated constraint $Y \perp\!\!\!\perp Q|W$ hits the core marked by the vertical violet rectangle. If for simplicity the weights are all constants, the remaining cores can be optimally hit by $X \perp\!\!\!\perp Y|Z,W$ (in bold). 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.

3.6. Imposing Acyclicity and Sparsity

Our approach also allows for integrating different structural search 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 $X \to Y$ 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 $X \to Y$ 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 implemented the branch-and-bound approach and all of the search techniques described in Section 3. The resulting open-source C++ implementation *bcause* of the approach is available at

`https://www.cs.helsinki.fi/group/coreo/bcause/`

In the following, we present results from an empirical evaluation of the running time performance of *bcause* on problem instances obtained from real-world datasets from several perspectives: (i) the marginal contribution of the different proposed search techniques, (ii) the efficiency of the approach with respect to current state of the art, and (iii) the impact of the scoring function used for obtaining constraint weights on the efficiency of the approach.

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) and also used in the original paper describing the *Dseptor* system (Hyttinen et al., 2017b); see Table 1 for more details on the benchmarks. As the basis of the causal discovery instances, we considered suitable-sized (n) subsets of the first non-constant variables in the datasets, making the remaining variables thus latent (recall that latent variables are supported by our general search space). This resulted in a total of 120 benchmark instances. For parts (i) and (ii) of the evaluation, we obtained the constraint weights by (local) Bayesian model selection with the BDeu (ESS=10, $\alpha = 0.5$) score[5]; further parameter values are considered in part (iii) of the evaluation. When reporting running times, we do not include the constraint weight computation times. We note that in this problem setting, the weight computation times are negligible to the running times of *Dseptor* or *bcause*. Concretely, obtaining the constraint weights for any single data set used in the

[5]This is the score used in the original paper describing *Dseptor* (Hyttinen et al., 2017b) and also in the preliminary version of this article (Rantanen et al., 2018).

experiments takes less than a second. The longest weight computation time was 0.7 seconds, on the Link10000 dataset.

For the experiments, we used the CPLEX system as the linear programming solver for obtaining core-based bounds within *bcause*. 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.

4.1. Impact of Search Techniques

We start the overview of the results by looking at the marginal contribution of the proposed individual search techniques as implemented in *bcause*. Here by marginal contribution we refer to switching off an individual search technique, instead applying a more basic ("baseline") version of the techniques as necessary, while keeping all search techniques intact. Concretely, we study the marginal contribution of three search techniques.

- The domain-specific branching heuristics detailed in Section 3.4. As a baseline heuristic, we compare to choosing the edge to branch on uniformly at random ("random branching").
- The inference rules 1–4 detailed in Section 3.3, allowing for inferring irrelevant edges under partial solutions. As a baseline comparison, we simply switch off the rules.
- The lower bounding technique detailed in Section 3.5 based on solving a linear relaxation of the minimum-cost hitting set problem over the unsatisfiable core patterns under partial solutions. As a baseline comparison, we switch off this additional bounding technique, and only obtain naive lower bounds by summing up the weights of the constraints that are known to be violated by the current partial solution.

The results are shown in Figures 9–11. Each plot gives a comparison of the per-instance running times of the default settings of *bcause* on the x-axis (with all three search techniques switched on) and *bcause* with one of the three techniques individually switched off on the y-axis. The shapes of points distinguish between the different numbers of random variables in the underlying datasets (excluding latent variables). As shown in Figure 9, the marginal contribution of the domain-specific branching heuristic is noticeable, as it yields considerable performance gains over using random branching, making the domain-specific branching heuristic integral for *bcause*. As shown in Figure 10, while their impact is more moderate, the inference rules 1–4 also consistently speed up search. (Recall that these rules do not have on impact on the number of search tree nodes visited, but rather lower the time spent at each search tree node.) Finally, as shown in Figure 11, the use of the core-based lower bounds obtained via linear programming also considerably speed up *bcause*, and most importantly very consistently for harder instances.

In summary, each of the three search techniques has a non-negligible marginal contribution to the performance of *bcause*, each consistently improving the running time performance of the approach.

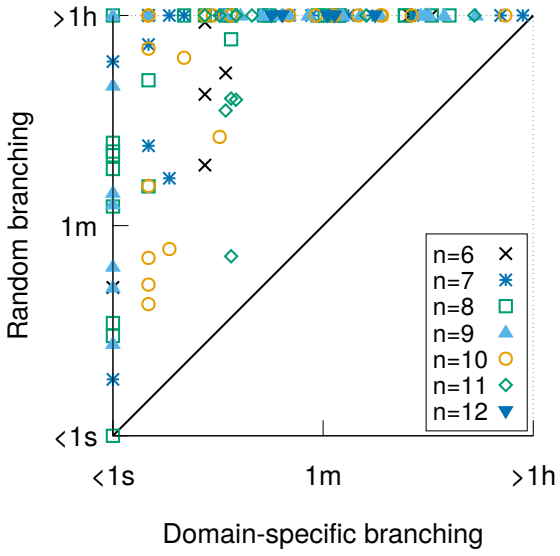

Figure 9: Marginal contribution of the domain-specific branching heuristic: per-instance running time comparison of *bcause* using domain-specific (x-axis) and random branching (y-axis).

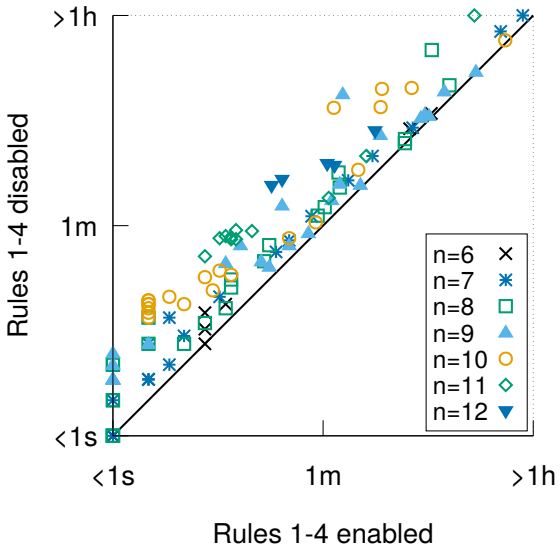

Figure 10: Marginal contribution of the irrelevant edge rules 1–4: per-instance running time comparison of *bcause* using (x-axis) and not using (y-axis) the rules.

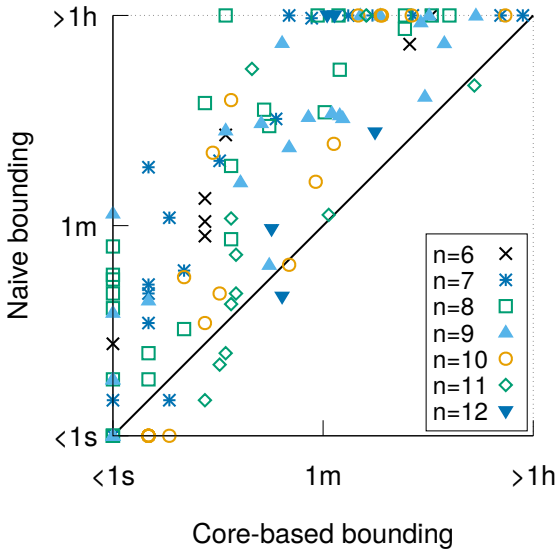

Figure 11: Marginal contribution of the unsatisfiable core linear programming lower bounds: per-instance running time comparison of *bcause* using (x-axis) and not using (y-axis) the core-based bounding.

4.2. Comparison with Current State of the Art

We compare the running time performance of *bcause* to that of *Dseptor* (Hyttinen et al., 2017b) as representative of the current state of the art. The *Dseptor* system is based on encoding of the causal discovery problem declaratively using the Boolean optimization paradigm of maximum satisfiability (MaxSAT), and furthermore integrates domain-specific techniques within a hybrid MaxSAT solver (Saikko et al., 2016a) making use of both SAT and integer programming solvers based on the so-called implicit hitting set paradigm (Moreno-Centeno and Karp, 2013; Davies and Bacchus, 2013; Saikko et al., 2016b).

Here we compare the performance of *bcause* and *Dseptor* in the general, unrestricted search space (allowing latent variables and cycles) as well as the restricted acyclic search space (will allowing latent variables). The results are shown in Figures 12 and 13, and Table 1. The plot in Figure 12 gives the number of instances solved (x-axis) by each approach under different per-instance time limits (y-axis); essentially, the further to the right the line, the better overall running time performance an approach exhibits. Evidently the performance of *bcause* is better in terms of the number of instances solved: within the 1-h per-instance time limit, *bcause* solved over 110 instance (both in the cyclic and acyclic case), while *Dseptor* solved less than 70 instance (both in the cyclic and acyclic case). Interestingly, the model space restriction has essentially no effect on the running times of *bcause*, while enforcing acyclicity degrades the performance of *Dseptor* slightly. As further seen in Figure 12, on a clear majority of the benchmarks *bcause* exhibits noticeably better runtimes than *Dseptor* re-

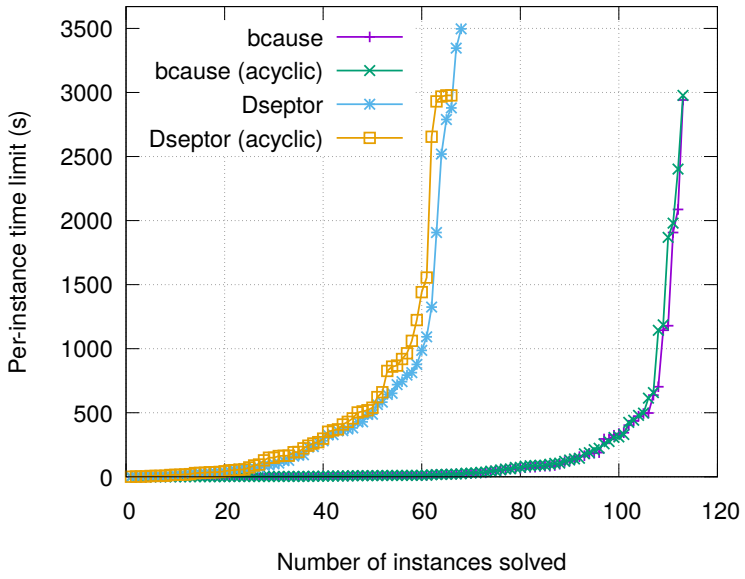

Figure 12: Comparison of *bcause* and *Dseptor*: number of solved instances (x-axis) for different per-instance time limits (y-axis), for both unrestricted search space and search space restricted to acyclic graphs.

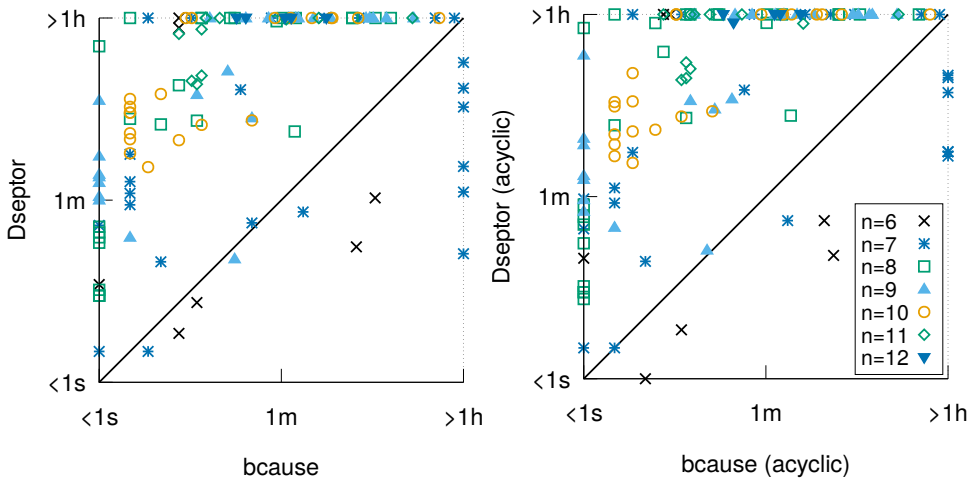

Figure 13: Comparison of *bcause* and *Dseptor*: per-instance running times for both unrestricted search space (left) and search space restricted to acyclic graphs (right).

Table 1: Running times of *bcause* and *Dseptor* over different search spaces.

		Running times (s)							
		General			Acyclic			Max-degree 3	
Dataset	n	***bcause***		***Dseptor***	***bcause***		***Dseptor***	***bcause***	
Adult	7	**1907**	(568)	>3600	**1980**	(380)	>3600	859	(121)
Alarm	9	**403**	(3)	>3600	**425**	(3)	>3600	406	(3)
Autos	9	**473**	(4)	>3600	**479**	(4)	>3600	447	(3)
Bands	8	**296**	(3)	>3600	**307**	(2)	>3600	276	(2)
Epigenetics	7	>3600	(>3600)	**18**	>3600	(>3600)	**165**	>3600	(>3600)
Flag	12	**75**	(30)	>3600	**83**	(32)	>3600	59	(15)
Heart	12	**27**	(24)	>3600	**29**	(25)	2977	16	(13)
Hepatitis	12	**65**	(27)	>3600	**73**	(29)	>3600	50	(13)
Horse.23	9	**83**	(4)	>3600	**90**	(4)	>3600	82	(3)
Horse	11	**11**	(7)	>3600	**12**	(7)	>3600	8	(4)
Image	8	**703**	(21)	>3600	**1869**	(46)	>3600	706	(43)
Imports	8	**295**	(1)	>3600	**311**	(1)	>3600	352	(1)
Letter	7	>3600	(>3600)	**487**	>3600	(>3600)	**622**	>3600	(>3600)
LungCancer	10	**184**	(10)	>3600	**212**	(11)	>3600	316	(15)
Meta	7	>3600	(>3600)	**128**	>3600	(>3600)	**149**	>3600	(>3600)
Mushroom1000	7	>3600	(>3600)	**1325**	>3600	(>3600)	**868**	>3600	(>3600)
Mushroom8124	7	>3600	(>3600)	**743**	>3600	(>3600)	**919**	>3600	(>3600)
Parkinsons	7	**344**	(27)	>3600	**257**	(20)	>3600	156	(14)
Sensors	7	>3600	(>3600)	**72**	>3600	(>3600)	**164**	>3600	(>3600)
Soybean	11	**10**	(8)	2789	**10**	(8)	1223	7	(5)
Spectf	11	**10**	(8)	987	**11**	(9)	1061	10	(6)
Statlog	7	**31**	(14)	36	**441**	(78)	>3600	31	(17)
SteelPlates	6	494	(460)	**63**	223	(191)	**35**	51	(25)
Voting	9	**637**	(7)	>3600	**614**	(7)	>3600	736	(7)
Water	9	**436**	(12)	>3600	**657**	(2)	>3600	486	(7)
Wdbc	8	**54**	(37)	3346	**61**	(42)	2977	55	(35)
Wine	9	**1179**	(9)	>3600	**1143**	(9)	>3600	1344	(10)
Zoo	7	**157**	(82)	>3600	**145**	(69)	>3600	152	(67)
alarm10000	10	>3600	(>3600)	>3600	>3600	(>3600)	>3600	2547	(34)
alarm1000	10	**119**	(37)	>3600	**107**	(22)	>3600	108	(7)
alarm100	9	**18**	(1)	1092	**19**	(1)	429	20	(1)
asia10000	8	**81**	(51)	281	**105**	(70)	370	180	(95)
asia1000	7	98	(<1)	**46**	98	(<1)	**35**	94	(29)
asia100	7	3	(<1)	**2**	2	(<1)	2	3	(<1)
carpo10000	11	**139**	(65)	>3600	**139**	(64)	2930	64	(28)
carpo1000	10	**2087**	(2)	>3600	**2402**	(2)	>3600	2221	(2)
carpo100	10	**52**	(1)	>3600	**54**	(1)	>3600	48	(1)
Diabetes10000	8	**<1**	(<1)	7	**<1**	(<1)	6	<1	(<1)
Diabetes1000	8	**<1**	(<1)	7	**<1**	(<1)	7	<1	(<1)
Diabetes100	9	**88**	(<1)	>3600	**90**	(<1)	>3600	103	(<1)
hailfinder10000	10	**2**	(2)	429	**2**	(2)	194	2	(1)
hailfinder1000	9	**1**	(<1)	65	**1**	(<1)	43	1	(<1)
hailfinder100	8	**498**	(<1)	>3600	**497**	(<1)	>3600	587	(<1)
insurance10000	9	**183**	(1)	>3600	**179**	(1)	>3600	206	(1)
insurance100	10	**337**	(4)	>3600	**330**	(3)	>3600	378	(3)
Link10000	12	**165**	(13)	>3600	**131**	(12)	>3600	187	(14)
Link1000	11	**1144**	(4)	>3600	**1186**	(4)	>3600	1362	(4)
Link100	10	**189**	(3)	>3600	**188**	(3)	>3600	214	(2)
Mildew10000	10	**4**	(3)	652	**3**	(2)	964	3	(2)
Mildew1000	8	**1**	(<1)	33	**1**	(<1)	38	1	(<1)
Mildew100	7	**2941**	(1247)	>3600	**2977**	(1258)	>3600	2738	(1021)
Pigs10000	10	**74**	(1)	>3600	**88**	(1)	>3600	85	(1)
Pigs1000	10	**7**	(3)	>3600	**8**	(3)	>3600	6	(2)
Pigs100	8	**<1**	(<1)	23	**<1**	(<1)	21	<1	(<1)
Water10000	9	**71**	(1)	>3600	**74**	(2)	>3600	78	(1)
Water1000	12	**22**	(20)	>3600	**23**	(21)	>3600	19	(17)
Water100	11	**15**	(8)	>3600	**17**	(9)	>3600	13	(6)

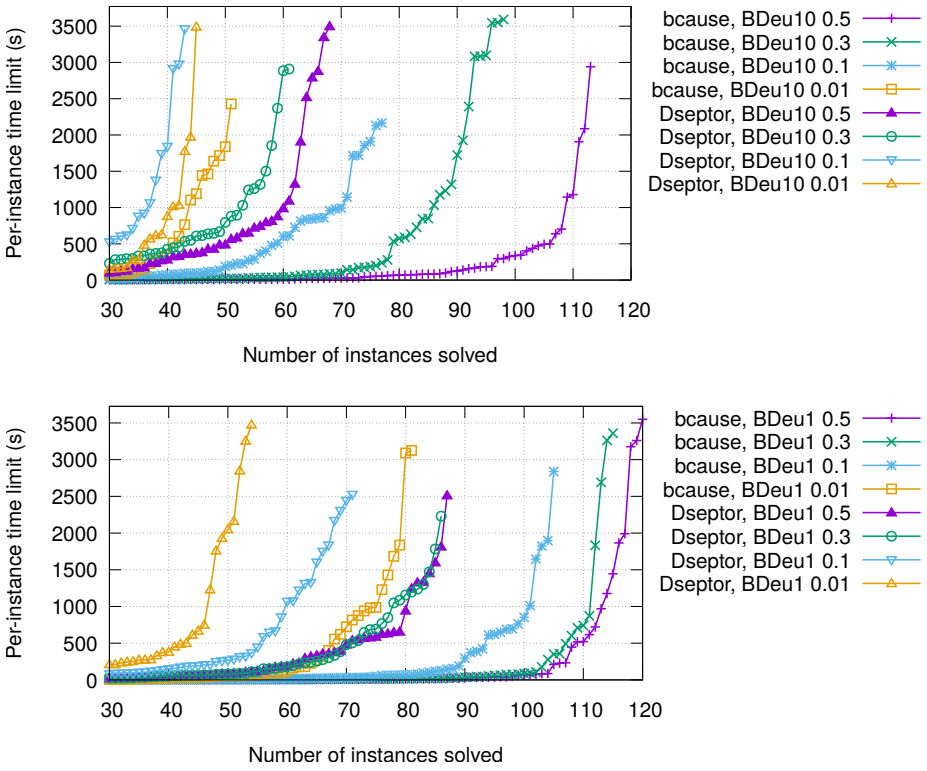

Figure 14: Comparison of *bcause* and *Dseptor*: number of solved instances (x-axis) for different per-instance time limits (y-axis), with constraint weights obtained by BDeu scoring using ESS 1 (top) and 10 (bottom) with prior probabilities of independence (α) as 0.5, 0.3, 0.1 and 0.01.

gardless of whether acyclicity is enforced, and times out less frequently, with 7 and 52 timeouts, respectively, without enforcing acyclicity, 7 and 54 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. The better running time between *bcause* and *Dseptor* for each instance and search spaces is given in bold. Apart from the fact that *bcause* quite consistently exhibits better running times than *Dseptor*, we also observe that *bcause* exhibits very good anytime performance in that it reaches an optimal solution often relatively fast.

4.3. Impact of Scoring Function Parameters

To further study the scalability of *bcause* and *Dseptor* and the impact of the scoring function parameters used to generated the causal discovery instances,

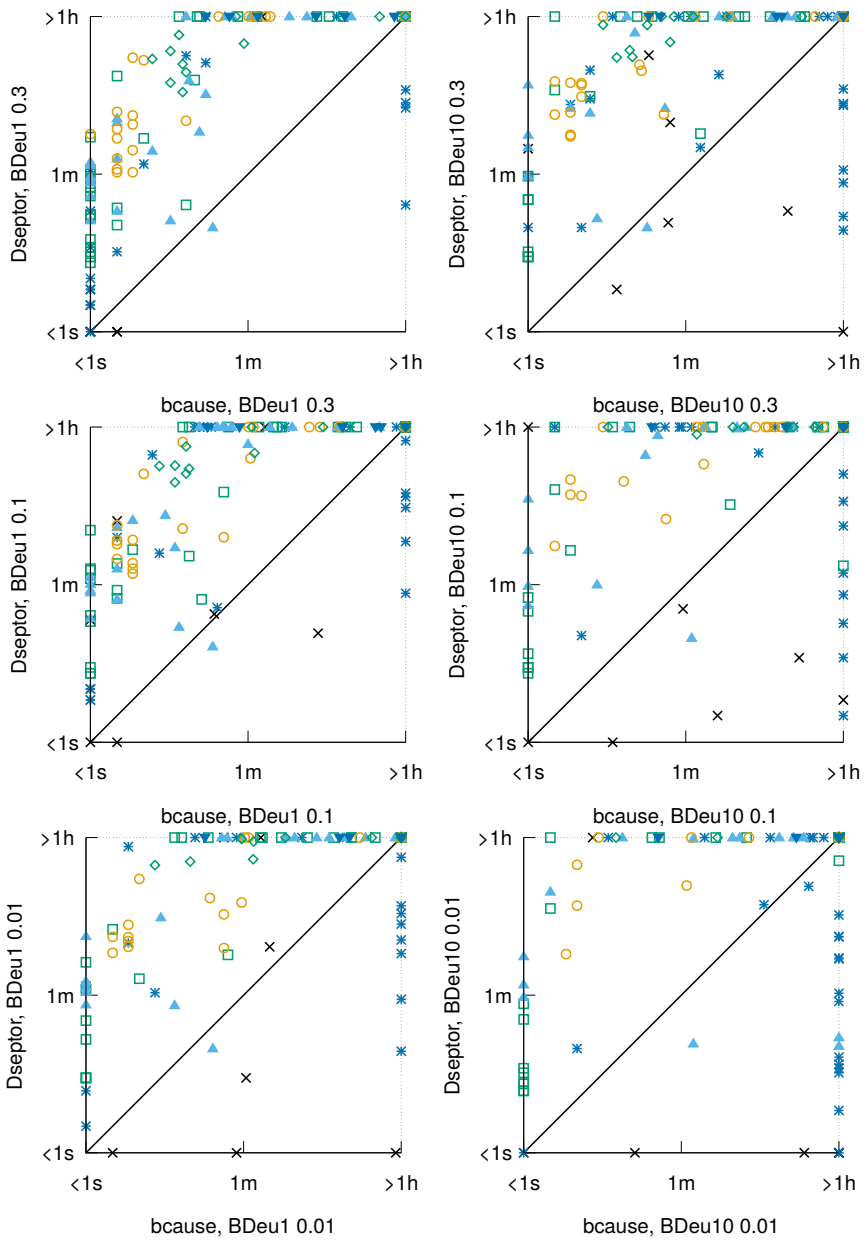

Figure 15: Comparison of *bcause* and *Dseptor*: per-instance running times with constraint weights obtained by BDeu scoring using ESS 1 (left column) and 10 (right column) and prior probabilities of independence as $\alpha = 0.3$ (top row), $\alpha = 0.1$ (middle row) and $\alpha = 0.01$ (bottom row).

we generated further problem instances based on the same datasets using BDeu scoring with equivalent sample sizes of 1 and 10, and prior probabilities of independence (α) 0.5, 0.3, 0.1 and 0.01.

As seen in Figure 14, for any fixed choice of scoring function parameters, *bcause* is able to solve more instances than *Dseptor*. A per-instance comparison for the individual parameter value pairs is shown in Figure 15. Interestingly, while *bcause* exhibits better overall performance than *Dseptor* on any choice of the parameters, the choice of scoring function parameters has a noticeable impact on the scalability of both *bcause* and *Dseptor*. We hypothesize this to originate from the intuition that lower prior probabilities of independence often result in more dependencies, which in turn may translate to many of the optimal graphs being denser. Considering *bcause*, the increased density may make the problem instances more difficult to solve in two ways. Firstly, the more independences there are, the easier it is for our independence-based heuristic to navigate in the search tree to the optimal solution. The second reason concerns the score-equivalent solutions in the overall search space, i.e., graphs that share the exact same weight. In particular, some causal graphs are able to satisfy the same set of (in)dependence constraints even after some edges are removed or reoriented (see Section 3.3). Intuitively, the more edges we assign within the search, the more equivalent solutions we are likely to encounter, which can to an extent be detrimental to search performance.

5. Related Work

Declarative Boolean satisfiability (SAT) solvers were first used in (Triantafillou et al., 2010; Triantafilou et al., 2010; Hyttinen et al., 2013) for developing approaches to discovering causal structures with latent confounders from several data sets in constraint-based fashion. Building on these ideas, Hyttinen et al. (2014) proposed the first exact approach to the problem setting considered in this article, in the form of a declarative framework in the language of answer set programming (ASP). 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 by Borboudakis and Tsamardinos (2016). Forré and Mooij (2018) apply a different separation condition for non-linear cyclic models. 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 by Hyttinen et al. (2017b), where 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.

Several inexact constraint-based algorithm are available for learning causal graphs under restricting assumptions. FCI learns acyclic graphs allowing for latent confounding and selection bias (Spirtes et al., 2000). CCD learns graphs with cycles (Richardson, 1996a). RFCI of Colombo et al. (2012) improves on efficiency of FCI. Claassen et al. (2013) developed a polynomial-time FCI-type

algorithm for discovering edge-minimal acyclic graphs with latent variables. Order independent versions of FCI and CCD by Colombo and Maathuis (2014) give more stable results in sparse high-dimensional settings.

Apart from these methods aiming for scalability, there are approaches focusing on accuracy. Conservative FCI of Ramsey et al. (2012) performs additional independence tests to not output orientations due to conflicting constraints. Claassen and Heskes (2012) combine weighted independence constraints with an inexact FCI-type procedure. Ogarrio et al. (2016) use FCI orientation rules to detect latent confounders over a skeleton obtained by a greedy score-based approach. Jabbari et al. (2017) use RFCI to find candidate PAGs fitting data and to generate a set of (in)dependence constraints K. Then an optimal graph is found among the candidates over an objective function consisting of weighted independence constraints in K.

For models with parametric restrictions, also score-based algorithms have been proposed. Evans and Richardson (2010) find maximum likelihood parameters for binary semi-Markovian models and Drton et al. (2009) for linear SEMs with correlated errors (both acyclic). Subsequently high-scoring graphs can be found with a greedy procedure e.g. using a BIC penalty (Evans and Richardson, 2010; Tsirlis et al., 2018).

In contrast to this related work, our procedure tackles a more general search space allowing for cycles and latent confounders and makes no parametric assumptions as such; it also offers guaranteed optimality of the solution. Our approach also straightforwardly generalizes to several data sets with partially overlapping variable sets (e.g. by simply combining weights for the constraints testable in several data sets) (Tillman and Spirtes, 2011; Tillman et al., 2008; Triantafillou et al., 2010; Hyttinen et al., 2014). Finally, we note that different branch-and-bound style algorithms have also been proposed for learning the structure of Bayesian networks (Suzuki, 1996; Tian, 2000; Malone and Yuan, 2014; van Beek and Hoffmann, 2015; Suzuki and Kawahara, 2017) and chordal Markov networks (Rantanen et al., 2017).

6. Conclusions

We developed a novel branch-and-bound approach to learning provably-optimal causal graphs in general search spaces. In contrast to the earlier approaches heavily relying on declarative optimization solvers, our approach is a specialized algorithm for the problem, and integrates knowledge about the problem domain for speeding up search via problem-specific branching heuristics; optimized algorithms for evaluating the objective function of the problem during search and inference rules for detecting which edges are irrelevant in terms of d-connectivity under a current partial solution; as well as integrating linear programming relaxation computations for lower bounding applicable during search based on problem-specific unsatisfiable cores. As we explained, the approach also allows for integrating search space restrictions, such as acyclicity or a degree bound, to the search. Through an extensive empirical evaluation, we showed that our open-source implementation *bcause* of the approach improves

on current state of the art in exact approaches to learning optimal causal graphs in terms of running times on real-world datasets. We foresee various direction for further work. For example, the approach could be modified to use different separation criteria, to account for phenomena such as selection bias, measurement noise and also data recorded in multiple different contexts. For runtime improvements, the approach currently does not make use of problem-specific symmetries or parallel computations. The potential of further search heuristics, including lookahead, could also be studied. Furthermore, the impact of dataset properties on the relative runtime performance of *Dseptor* and *bcause* could yield further insights into which types of search techniques result in improved performance on individual datasets.

Acknowledgments

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

References

Abellán, J., Gómez-Olmedo, M., Moral, S., 2006. Some variations on the PC algorithm, in: Studený, M., Vomlel, J. (Eds.), Third European Workshop on Probabilistic Graphical Models, 12-15 September 2006, Prague, Czech Republic. Electronic Proceedings., pp. 1–8.

Bartlett, M., Cussens, J., 2017. Integer linear programming for the Bayesian network structure learning problem. Artificial Intelligence 244, 258–271.

van Beek, P., Hoffmann, H., 2015. Machine learning of Bayesian networks using constraint programming, in: Pesant, G. (Ed.), Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings, Springer. pp. 429–445.

Biere, A., Heule, M., van Maaren, H., Walsh, T. (Eds.), 2009. Handbook of Satisfiability. volume 185 of *Frontiers in Artificial Intelligence and Applications*, IOS Press.

Bollen, K.A., 1989. Structural Equations with Latent Variables. John Wiley & Sons.

Borboudakis, G., Tsamardinos, I., 2016. Towards robust and versatile causal discovery for business applications, in: Krishnapuram, B., Shah, M., Smola, A.J., Aggarwal, C.C., Shen, D., Rastogi, R. (Eds.), Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, ACM. pp. 1435–1444.

Buntine, W.L., 1991. Theory refinement on Bayesian networks, in: D'Ambrosio, B., Smets, P. (Eds.), UAI '91: Proceedings of the Seventh Annual Conference on Uncertainty in Artificial Intelligence, University of California at Los

Angeles, Los Angeles, CA, USA, July 13-15, 1991, Morgan Kaufmann. pp. 52–60.

Claassen, T., Heskes, T., 2012. A Bayesian approach to constraint based causal inference, in: de Freitas, N., Murphy, K.P. (Eds.), Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012, AUAI Press. pp. 207–216.

Claassen, T., Mooij, J.M., Heskes, T., 2013. Learning sparse causal models is not NP-hard, in: Nicholson, A., Smyth, P. (Eds.), Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013, AUAI Press. pp. 172–181.

Colombo, D., Maathuis, M.H., 2014. Order-independent constraint-based causal structure learning. Journal of Machine Learning Research 15, 3741–3782.

Colombo, D., Maathuis, M.H., Kalisch, M., Richardson, T.S., 2012. Learning high-dimensional directed acyclic graphs with latent and selection variables. Annals of Statistics 40, 294–321.

Cooper, G.F., 1997. A simple constraint-based algorithm for efficiently mining observational databases for causal relationships. Data Min. Knowl. Discov. 1, 203–224.

Cooper, G.F., Herskovits, E., 1992. A Bayesian method for the induction of probabilistic networks from data. Machine Learning 9, 309–347.

Davies, J., Bacchus, F., 2013. Exploiting the power of MIP solvers in MAXSAT, in: Järvisalo, M., Gelder, A.V. (Eds.), Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings, Springer. pp. 166–181.

Drton, M., Eichler, M., Richardson, T.S., 2009. Computing maximum likelihood estimates in recursive linear models with correlated errors. Journal of Machine Learning Research 10, 2329–2348.

Evans, R.J., Richardson, T.S., 2010. Maximum likelihood fitting of acyclic directed mixed graphs to binary data, in: Grünwald, P., Spirtes, P. (Eds.), UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8-11, 2010, AUAI Press. pp. 177–184.

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

Forré, P., Mooij, J.M., 2018. Constraint-based causal discovery for non-linear structural causal models with cycles and latent confounders, in: Globerson, A., Silva, R. (Eds.), Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018, AUAI Press. pp. 269–278.

Geiger, D., Heckerman, D., 2002. Parameter priors for directed acyclic graphical models and the characterization of several probability distributions. The Annals of Statistics 30, 1412–1440.

Heckerman, D., Geiger, D., Chickering, D.M., 1995. Learning Bayesian networks: The combination of knowledge and statistical data. Machine Learning 20, 197–243.

Hyttinen, A., Eberhardt, F., Hoyer, P.O., 2012. Learning linear cyclic causal models with latent variables. Journal of Machine Learning Research 13, 3387–3439.

Hyttinen, A., Eberhardt, F., Järvisalo, M., 2014. Constraint-based causal discovery: Conflict resolution with answer set programming, in: Zhang, N.L., Tian, J. (Eds.), Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI 2014, Quebec City, Quebec, Canada, July 23-27, 2014, AUAI Press. pp. 340–349.

Hyttinen, A., Hoyer, P.O., Eberhardt, F., Järvisalo, M., 2013. Discovering cyclic causal models with latent variables: A general sat-based procedure, in: Nicholson, A., Smyth, P. (Eds.), Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013, AUAI Press. pp. 301–310.

Hyttinen, A., Plis, S.M., Järvisalo, M., Eberhardt, F., Danks, D., 2017a. A constraint optimization approach to causal discovery from subsampled time series data. International Journal of Approximate Reasoning 90, 208–225.

Hyttinen, A., Saikko, P., Järvisalo, M., 2017b. A core-guided approach to learning optimal causal graphs, in: Sierra, C. (Ed.), Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, ijcai.org. pp. 645–651.

Jabbari, F., Ramsey, J., Spirtes, P., Cooper, G.F., 2017. Discovery of causal models that contain latent variables through Bayesian scoring of independence constraints, in: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18-22, 2017, Proceedings, Part II, Springer. pp. 142–157.

Lacerda, G., Spirtes, P., Ramsey, J., Hoyer, P.O., 2008. Discovering cyclic causal models by independent components analysis, in: McAllester, D.A., Myllymäki, P. (Eds.), UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008, AUAI Press. pp. 366–374.

Magliacane, S., Claassen, T., Mooij, J.M., 2016. Ancestral causal inference, in: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, Curran Associates, Inc.. pp. 4466–4474.

Malone, B., Järvisalo, M., Myllymäki, P., 2015. Impact of learning strategies on the quality of Bayesian networks: An empirical evaluation, in: Meila, M., Heskes, T. (Eds.), Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands, AUAI Press. pp. 562–571.

Malone, B.M., Yuan, C., 2014. A depth-first branch and bound algorithm for learning optimal Bayesian networks, in: Croitoru, M., Rudolph, S., Woltran, S., Gonzales, C. (Eds.), Graph Structures for Knowledge Representation and Reasoning - Third International Workshop, GKR 2013, Beijing, China, August 3, 2013. Revised Selected Papers, Springer. pp. 111–122.

Margaritis, D., Bromberg, F., 2009. Efficient Markov network discovery using particle filters. Computational Intelligence 25, 367–394.

Moreno-Centeno, E., Karp, R.M., 2013. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. Operations Research 61, 453–468.

Natori, K., Uto, M., Ueno, M., 2017. Consistent learning bayesian networks with thousands of variables, in: Hyttinen, A., Suzuki, J., Malone, B.M. (Eds.), Proceedings of the 3rd Workshop on Advanced Methodologies for Bayesian Networks, AMBN 2017, Kyoto, Japan, September 20-22, 2017, PMLR. pp. 57–68.

Neal, R., 2000. On deducing conditional independence from d-separation in causal graphs with feedback. Journal of Artificial Intelligence Research 12, 87–91.

Ogarrio, J.M., Spirtes, P., Ramsey, J., 2016. A hybrid causal search algorithm for latent variable models, in: Antonucci, A., Corani, G., de Campos, C.P. (Eds.), Probabilistic Graphical Models - Eighth International Conference, PGM 2016, Lugano, Switzerland, September 6-9, 2016. Proceedings, JMLR.org. pp. 368–379.

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

Pearl, J., Dechter, R., 1996. Identifying independencies in causal graphs with feedback, in: Horvitz, E., Jensen, F.V. (Eds.), UAI '96: Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence, Reed College, Portland, Oregon, USA, August 1-4, 1996, Morgan Kaufmann. pp. 420–426.

Ramsey, J., Zhang, J., Spirtes, P., 2012. Adjacency-faithfulness and conservative causal inference. CoRR abs/1206.6843.

Rantanen, K., Hyttinen, A., Järvisalo, M., 2017. Learning chordal markov networks via branch and bound, in: Guyon, I., von Luxburg, U., Bengio,

S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA, pp. 1845–1855.

Rantanen, K., Hyttinen, A., Järvisalo, M., 2018. Learning optimal causal graphs with exact search, in: Studený, M., Kratochvíl, V. (Eds.), International Conference on Probabilistic Graphical Models, PGM 2018, 11-14 September 2018, Prague, Czech Republic, PMLR. pp. 344–355.

Richardson, T., 1996a. A discovery algorithm for directed cyclic graphs, in: Horvitz, E., Jensen, F.V. (Eds.), UAI '96: Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence, Reed College, Portland, Oregon, USA, August 1-4, 1996, Morgan Kaufmann. pp. 454–461.

Richardson, T.S., 1996b. Feedback Models: Interpretation and Discovery. Ph.D. thesis. Carnegie Mellon University.

Saikko, P., Berg, J., Järvisalo, M., 2016a. LMHS: A SAT-IP hybrid MaxSAT solver, in: Creignou, N., Berre, D.L. (Eds.), Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings, Springer. pp. 539–546.

Saikko, P., Wallner, J.P., Järvisalo, M., 2016b. Implicit hitting set algorithms for reasoning beyond NP, in: Baral, C., Delgrande, J.P., Wolter, F. (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016., AAAI Press. pp. 104–113.

Shachter, R.D., 1998. Bayes-Ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams), in: Cooper, G.F., Moral, S. (Eds.), UAI '98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, University of Wisconsin Business School, Madison, Wisconsin, USA, July 24-26, 1998, Morgan Kaufmann. pp. 480–487.

Spirtes, P., 1995. Directed cyclic graphical representations of feedback models, in: Besnard, P., Hanks, S. (Eds.), UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, Montreal, Quebec, Canada, August 18-20, 1995, Morgan Kaufmann. pp. 491–498.

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

Steck, H., Jaakkola, T.S., 2002. On the dirichlet prior and bayesian regularization, in: Becker, S., Thrun, S., Obermayer, K. (Eds.), Advances in Neural Information Processing Systems 15 [Neural Information Processing Systems, NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada], MIT Press. pp. 697–704.

Studený, M., 1998. Bayesian networks from the point of view of chain graphs, in: Cooper, G.F., Moral, S. (Eds.), UAI '98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, University of Wisconsin Business School, Madison, Wisconsin, USA, July 24-26, 1998, Morgan Kaufmann. pp. 496–503.

Suzuki, J., 1996. Learning Bayesian belief networks based on the minimum description length principle: An efficient algorithm using the B & B technique, in: Saitta, L. (Ed.), Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996, Morgan Kaufmann. pp. 462–470.

Suzuki, J., Kawahara, J., 2017. Branch and bound for regular Bayesian network structure learning, in: Elidan, G., Kersting, K., Ihler, A.T. (Eds.), Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017, AUAI Press.

Tian, J., 2000. A branch-and-bound algorithm for MDL learning Bayesian networks, in: Boutilier, C., Goldszmidt, M. (Eds.), UAI '00: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, Stanford University, Stanford, California, USA, June 30 - July 3, 2000, Morgan Kaufmann. pp. 580–588.

Tillman, R.E., Danks, D., Glymour, C., 2008. Integrating locally learned causal structures with overlapping variables, in: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (Eds.), Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008, Curran Associates, Inc.. pp. 1665–1672.

Tillman, R.E., Spirtes, P., 2011. Learning equivalence classes of acyclic models with latent and selection variables from multiple datasets with overlapping variables, in: Gordon, G.J., Dunson, D.B., Dudík, M. (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011, JMLR.org. pp. 3–15.

Triantafillou, S., Tsamardinos, I., Tollis, I.G., 2010. Learning causal structure from overlapping variable sets, in: AISTATS, JMLR. pp. 860–867.

Triantafilou, S., Tsamardinos, I., Tollis, I.G., 2010. Learning causal structure from overlapping variable sets, in: Teh, Y.W., Titterington, D.M. (Eds.), Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010, JMLR.org. pp. 860–867.

Tsirlis, K., Lagani, V., Triantafillou, S., Tsamardinos, I., 2018. On scoring maximal ancestral graphs with the max-min hill climbing algorithm. International Journal on Approximate Reasoning 102, 74–85.

Verma, T., Pearl, J., 1990. Equivalence and synthesis of causal models, in: Bonissone, P.P., Henrion, M., Kanal, L.N., Lemmer, J.F. (Eds.), UAI '90: Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence, MIT, Cambridge, MA, USA, July 27-29, 1990, Elsevier. pp. 255–270.

Wright, S., 1934. The method of path coefficients. Annals of Mathematical Statistics 5, 161–215.

Yuan, C., Malone, B., 2013. Learning optimal Bayesian networks: A shortest path perspective. Journal of Artificial Intelligence Research 48, 23–65.

Zhalama, Zhang, J., Eberhardt, F., Mayer, W., 2017. SAT-based causal discovery under weaker assumptions, in: Elidan, G., Kersting, K., Ihler, A.T. (Eds.), Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017, AUAI Press.