Learning Optimal Chain Graphs with Answer Set Programming

Dag Sonntag
ADIT, IDA,
Linköping University

Matti Järvisalo
HIIT, Dept. Comp. Sci.,
University of Helsinki

Jose M. Peña
ADIT, IDA,
Linköping University

Antti Hyttinen
HIIT, Dept. Comp. Sci.,
University of Helsinki

Abstract

Learning an optimal chain graph from data is an important hard computational problem. We present a new approach to solve this problem for various objective functions without making any assumption on the probability distribution at hand. Our approach is based on encoding the learning problem declaratively using the answer set programming (ASP) paradigm. Empirical results show that our approach provides at least as accurate solutions as the best solutions provided by the existing algorithms, and overall provides better accuracy than any single previous algorithm.

1 INTRODUCTION

Learning an optimal structure for a graphical model is a well-known and important hard computational problem. Indeed, various learning algorithms have been proposed over the years for different classes of graphical models. These algorithms can be categorized into in-exact (often local search style) approaches and exact approaches. The algorithms in the former category typically scale better, but are not in general guaranteed to produce optimal solutions without restrictive assumptions on the probability distribution at hand. The algorithms in the latter category, due to the NP-hardness of the underlying optimization problem, require more computational resources, but in turn can provide optimal solutions in much more general settings.

Two important, widely studied and applied classes of probabilistic graphical models are Bayesian networks (whose structure is represented by directed acyclic graphs), and Markov networks (represented by undirected graphs). In this paper we focus on *chain graphs* (CGs), a superclass of both Bayesian and Markov networks. CGs are hybrid graphs that can contain both directed and undirected edges, but are not allowed to contain semidirected cycles. This makes CGs more expressive than Bayesian and Markov networks in the sense that CGs can model both symmetric (like Markov networks) and asymmetric (like Bayesian networks) relations between random variables, and hence allow for a wider range of independence models to be represented. For example, for 20 random variables, CGs can represent approximately 1000 times as many models as Bayesian networks (Sonntag et al., 2015). There exist multiple interpretations of CGs in the literature. Here we focus on the classical LWF interpretation by Frydenberg (1990); Lauritzen and Wermuth (1989).

In this work we take on the challenging task of developing exact structure learning algorithms for CGs. To our best knowledge, to date only three learning algorithms CGs have been proposed: PC (Studený, 1997), LCD (Ma et al., 2008), and CKES (Peña et al., 2014). Each of these algorithms implement forms of local search, and are guaranteed to find (inclusion) optimal solutions only under restrictive assumptions on the input. Specifically, PC and LCD assume that the probability distribution at hand is faithful to a CG. The CKES algorithm, on the other hand, assumes that the probability distribution at hand satisfies the so-called composition property. While this assumption is considerably weaker than the faithfulness assumption, it leaves room for developing CG learning algorithms that can provide optimal solutions in more general settings.

The main contribution of this paper is a versatile approach to learning CGs. Our approach improves on the earlier CG learning algorithms in that it is guaranteed to find (e.g., inclusion) optimal CGs without making assumptions on the probability distribution at hand. Moreover, our approach can easily be adapted to produce optimal CGs wrt other objective functions such as CGs that represent the largest number of independencies, or CGs with the least number of edges. Our approach is based on encoding the CG learning problem in a modular way using the answer set programming (ASP) paradigm. This enables the use of recent advances in state-of-the-art exact ASP optimization solvers that allow for a complete search in the space of CGs. Moreover, due to the expressive constraint modelling

language offered by ASP, our approach allows for integrating user knowledge into the search. This can, for example, be deployed to enforce certain substructures in the produced solutions or to adopt advanced objective functions taking non-binary information about independence constraints into account.

We also present results from an empirical comparison of our approach with the existing CG learning algorithms. As the existing algorithms make assumptions about the probability distribution at hand, the experiments involve only probability distributions that satisfy the assumptions of the existing algorithms (although unnecessary for our approach) in order to avoid biasing the results in our favour. Even in such restricted settings, the results of the evaluation indicate that our approach provides at least as accurate solutions as the best solutions provided by the existing algorithms, and overall provides better accuracy than any single previous algorithm.

This work is motivated by recent work on harnessing declarative programming (including ASP, Boolean satisfiability, maximum satisfiability, and integer programming solvers) to learn optimal Bayesian networks (Cussens and Bartlett, 2013; Berg et al., 2014; Parviainen et al., 2014), Markov networkss (Corander et al., 2013) and causal structures (Hyttinen et al., 2013, 2014). However, to our best knowledge, this work is the first investigation into learning optimal CGs with declarative programming.

The rest of this paper is organized as follows. We start by reviewing CGs, their semantics, and the three existing CG learning algorithms (Section 2). We then continue by presenting and extending a set of inference rules for computing separations and non-separations in CGs (Section 3). In Section 4 we present an implementation of the CG learning approach, using the rules from the previous section. This implementation is then evaluated wrt the existing CG learning algorithms in Section 5.

2 CHAIN GRAPHS

In this section, we review concepts related to chain graphs as relevant to this work.

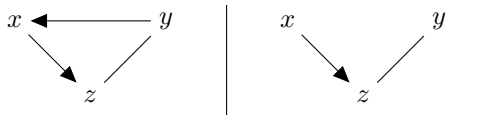
Unless otherwise stated, all the graphs in this paper are defined over a finite set of N nodes. Moreover, the graphs are *simple*, i.e., contain at most one edge between any pair of nodes. The elements of N are not distinguished from singletons. We consider graphs which may contain both undirected and directed edges. For a given graph G, we use $x_1 - x_2$ (resp., $x_1 \to x_2$) to denote that G contains an undirected (resp., directed edge) between two nodes x_1 and x_2.

The *skeleton* of G is the undirected graph that has the same adjacencies as G. A *route* between a node x_1 and a node x_n in G is a sequence of (not necessarily distinct) nodes $x_1, \ldots, x_n$ such that $x_i \to x_{i+1}$, $x_i \leftarrow x_{i+1}$, or $x_i - x_{i+1}$ for all $1 \leq i < n$. A route $x_1, \ldots, x_n$ in G is a *semidirected cycle* if (i) $x_n = x_1$, (ii) $x_1 \to x_2$ is in G, and (iii) $x_i \to x_{i+1}$ or $x_i - x_{i+1}$ is in G for all $1 < i < n$. A *chain graph* (CG) is a graph that contains no semidirected cycles.

A *section* of a route ρ in a CG is a *maximal* (wrt set inclusion) undirected subroute of ρ. A section $x_2 - \ldots - x_{n-1}$ of ρ is a *collider section* of ρ if $x_1 \to x_2 - \ldots - x_{n-1} \leftarrow x_n$ is a subroute of ρ. Moreover, ρ is *Z-open* with $Z \subseteq N$ if (i) every collider section of ρ has a node in Z, and (ii) no non-collider section of ρ has a node in Z. Let X, Y and Z denote three disjoint subsets of N. If there is no Z-open route in a CG G between nodes in X and nodes in Y, X is *separated* from Y given Z in G, denoted by $X \perp\!\!\!\perp_G Y | Z$, meaning that X and Y are represented as conditionally independent given Z in G. Otherwise, X is *non-separated* from Y given Z in G, denoted by $X \not\perp\!\!\!\perp_G Y | Z$. An independence model M is a set of statements of the form $X \perp\!\!\!\perp_M Y | Z$, meaning that X is independent of Y given Z. The *independence model* represented by G, denoted by $I(G)$, is the set of represented independencies $X \perp\!\!\!\perp_G Y | Z$. We denote by $X \perp\!\!\!\perp_p Y | Z$ (resp. $X \not\perp\!\!\!\perp_p Y | Z$) that X is independent (resp. dependent) of Y given Z under a probability distribution p. The independence model induced by p, denoted by $I(p)$, is the set of statements $X \perp\!\!\!\perp_p Y | Z$. A distribution p is *faithful* to a CG G when $X \perp\!\!\!\perp_p Y | Z$ iff $X \perp\!\!\!\perp_G Y | Z$ for all pairwise disjoint subsets X, Y, Z of N. Two CGs are *Markov equivalent* if they represent the same independence model.

Let X, Y, Z and W denote four disjoint subsets of N. An independence model M is a *semi-graphoid* if it has the following properties: (i) *symmetry*: $X \perp\!\!\!\perp_M Y | Z$ implies $Y \perp\!\!\!\perp_M X | Z$; (ii) *decomposition*: $X \perp\!\!\!\perp_M Y \cup W | Z$ implies $X \perp\!\!\!\perp_M Y | Z$; (iii) *weak union*: $X \perp\!\!\!\perp_M Y \cup W | Z$ implies $X \perp\!\!\!\perp_M Y | Z \cup W$; and (iv) *contraction*: $X \perp\!\!\!\perp_M Y | Z \cup W$ and $X \perp\!\!\!\perp_M W | Z$ together imply $X \perp\!\!\!\perp_M Y \cup W | Z$. A semi-graphoid M is a *graphoid* if it has the property (v) *intersection*: $X \perp\!\!\!\perp_M Y | Z \cup W$ and $X \perp\!\!\!\perp_M W | Z \cup Y$ together imply $X \perp\!\!\!\perp_M Y \cup W | Z$. A graphoid M is *compositional* if $X \perp\!\!\!\perp_M Y | Z$ and $X \perp\!\!\!\perp_M W | Z$ together imply $X \perp\!\!\!\perp_M Y \cup W | Z$. The independence model induced by a probability distribution is a semi-graphoid, and the independence model induced by a strictly positive probability distribution is a graphoid (Studený, 2005). While the independence model induced by a probability distribution is not a compositional graphoid in general, independence models induced by Gaussian probability distributions are compositional (Studený, 2005).

The *elementary statements* of a semi-graphoid M are those of the form $x \perp\!\!\!\perp_M y | Z$, where $x, y \in N$. A semi-graphoid M is determined by its elementary statements, meaning that every statement in M follows from the elementary statements by repeatedly applying the semi-graphoid properties (Studený, 2005, Lemma 2.2). Thus one can equiv-



(a) A graph G

(b) A CG H

$x \perp\!\!\!\perp_H y|z$
$y \perp\!\!\!\perp_H x|z$

(c) The independence model of H

$x \perp\!\!\!\perp_M y|\emptyset$
$y \perp\!\!\!\perp_M x|\emptyset$
$x \perp\!\!\!\perp_M y|z$
$y \perp\!\!\!\perp_M x|z$

(d) Another independence model M

Figure 1: Example

alently work with the elementary statements of a semigraphoid. We do so in the rest of the article. In other words, we restrict the independence model represented by a CG or induced by a probability distribution to its elementary statements.

A CG G is *inclusion optimal* wrt an independence model M if (i) $I(G) \subseteq M$, and (ii) there exists no CG H such that $I(G) \subset I(H) \subseteq M$. We say that an inclusion optimal G is *independence optimal* if there exists no CG H such that $I(H) \subseteq M$ and $|I(H)| > |I(G)|$.

Example 1 *Although the graph G shown in Figure 1a only contains directed and undirected edges, it is not a CG since it contains the semidirected cycle $x \to z - y \to x$. The graph H in Figure 1b is a CG; its independence model $I(H)$ shown in Figure 1c. For example, $x \perp\!\!\!\perp_H y|z$ holds because the only route between x and y is $x \to z - y$ and z is in the conditioning set. Moreover, $I(H)$ fulfills the graphoid properties since it is symmetric and only contains one pair of independencies. Finally, $I(H) \subset I(M)$ holds for the independence model M shown in Figure 1d. This means that H is inclusion optimal wrt M, since removing an edge from H would cause either $x \perp\!\!\!\perp_H z|\emptyset$ or $y \perp\!\!\!\perp_H z|\emptyset$ to be true, neither of which are in M. Also note that M is not faithful to any CG since there is no CG H' such that $I(H') = I(M)$.*

2.1 Algorithms for Learning CGs

We continue by a short overview of existing algorithms for CG learning. To our knowledge, only three such have been proposed earlier.

The first algorithm, proposed in (Studený, 1997), is a PC-like algorithm that first finds a skeleton, and then orients the edges according to a specific set of rules. The second algorithm, LCD, proposed by (Ma et al., 2008), uses a divide-and-conquer approach that allows for fast learning of CGs especially in the space of sparse CGs. If the probability distribution at hand is faithful to a CG G, then both Studený's algorithm and LCD are guaranteed to find a CG H that is Markov equivalent with G.

The third algorithm for learning CGs, CKES, proposed in (Peña et al., 2014), can be viewed as an extension of the KES (Nielsen et al., 2003) Bayesian network learning algorithm. CKES works by iteratively adding (resp. removing) edges between the variables in the CG that are dependent (resp. independent) in the probability distribution at hand. The CKES algorithm guarantees to produce a CG that is inclusion optimal wrt the independence model induced by the probability distribution at hand in case the model is a compositional graphoid.

3 INFERENCE RULES FOR FINDING SEPARATIONS IN CHAIN GRAPHS

In this work, we rely on inference rules by Studený (1998) that allow for efficiently computing the separations and non-separations in a given CG G. We will represent these rules in our ASP encoding of CG learning, as detailed in Section 4. The rules are based on four sets of nodes U_x^C, V_x^C, W_x^C, and Z_x^C, that are saturated for each node $x \in N$ wrt the conditioning set $C \subseteq N \setminus \{x\}$. A node x is then separated from each node $y \in N \setminus (U_x^C \cup V_x^C)$ given the conditioning set C, i.e., $x \perp\!\!\!\perp_G N \setminus (U_x^C \cup V_x^C)|C$. The rules are shown in Figure 2. For some intuition, note that

- $y \in V_x^C$ iff there exists a C-open route from x to y in G which contains the subroute $a_i \to a_{i+1} - \cdots - a_{i+k} = y$, $k \geq 1$,
- $y \in U_x^C$ iff there exists a C-open route from x to y in G which does not contain the subroute $a_i \to a_{i+1} - \cdots - a_{i+k} = y$, $k \geq 1$,
- $y \in W_x^C$ iff there exists a $z \in U_x^C \cup V_x^C$ and a route $z = a_0 \to a_1 - \cdots - a_r = y$ in G, $r \geq 1$, and
- $y \in Z_x^C$ iff there exists a $z \in U_x^C \cup V_x^C$ and a route $z = a_0 \to a_1 - \cdots - a_r = y$, $r \geq 1$, with $\{a_1, \ldots, a_r\} \cap C \neq \emptyset$.

Note that rule 7 in Figure 2 corrects a small typo in the original paper (Studený, 1998).

The correctness of these rules can be stated as follows.

Theorem 1 (Adapted from (Studený, 1998))
Given a CG G over a node set N, starting with $U_x^C = V_x^C = Z_x^C = W_x^C = \emptyset$ for each node $x \in N$ and $C \subseteq N$, apply the rules in Figure 2 until fixpoint. Then $y \notin U_x^C \cup V_x^C$ iff $x \perp\!\!\!\perp_G y|C$.

0. $C \subset N, x \notin C \Rightarrow x \in U_x^C$
1. $x \in U_a^C, x - y, y \notin C \Rightarrow y \in U_a^C$
2. $x \in U_a^C, y \to x, y \notin C \Rightarrow y \in U_a^C$
3. $x \in U_a^C \cup V_a^C, x \to y, y \notin C \Rightarrow y \in V_a^C$
4. $x \in V_a^C, x - y, y \notin C \Rightarrow y \in V_a^C$
5. $x \in U_a^C \cup V_a^C, x \to y \Rightarrow y \in W_a^C$
6. $x \in W_a^C, x - y \Rightarrow y \in W_a^C$
7. $x \in W_a^C, x \in C \Rightarrow x \in Z_a^C$
8. $x \in Z_a^C, x - y \Rightarrow y \in Z_a^C$
9. $x \in Z_a^C, y \to x, y \notin C \Rightarrow y \in U_a^C$

Figure 2: Studený's inference rules

3.1 Refining Studený's Rules

It turns out that the W sets in Studený's original rules are actually redundant. In detail, the W sets can be removed by replacing the original rules 5–7 by the rules 10–11 presented in Figure 3.

10. $x \in U_a^C \cup V_a^C, x \to y, y \in C \Rightarrow y \in Z_a^C$
11. $x \in V_a^C, x - y, y \in C \Rightarrow y \in Z_a^C$

Figure 3: Replacement for Studený's rules 5–7

Proposition 1 *Given a CG G over a node set N, starting with $U_x^C = V_x^C = Z_x^C = W_x^C = \emptyset$ for each node $x \in N$ and $C \subseteq N$, the following computations give the same U_x^C and V_x^C sets:*

- *Apply the rules 0–9 in Figure 2 until fixpoint.*
- *Apply rules 0–4 and rules 8–9 in Figure 2 together with rules 10–11 in Figure 3 until fixpoint.*

Proof (sketch). Without loss of generality, perform the original and the refined computation by running the following three steps repeatedly until fixpoint. Step 1: Run rules 0-4 until fixpoint. Step 2: Run rules 5-8 (original computation) or rules 8,10-11 (refined computation) until fixpoint. Step 3: Run rule 9 until fixpoint. In order to prove the lemma, it suffices to prove that Z_x^C is the same at the end of step 2 for both computations.

Consider arbitrary U_x^C and V_x^C. Recall that by using the original rules 0-9, a node $y \in Z_x^C$ only if there exists a node $z \in U_x^C \cup V_x^C$ and a route $z = a_0 \to a_1 - \cdots - a_r = y$, $r \geq 1$, with $\{a_1, \ldots, a_r\} \cap C \neq \emptyset$. Let a_s denote the first node in the subroute $a_1 - \cdots - a_r$ that is in C. Observe that $a_{s-1} \in U_x^C \cup V_x^C$ if $s = 1$, and $a_{s-1} \in V_x^C$ otherwise. One can check that we have exactly $a_s \in Z_x^C$ both by applying rules 10-11 in the refined computation and by applying rules 5-7 in the original computation. After this, $y \in Z_x^C$ is obtained both in the original and in the refined computation by repeatedly applying rule 8. □

3.2 Additional Rules

On top of the "replacement" rules 10–11, we further identify additional rules that can be applied soundly together with Studený's original rules, i.e., without affecting the resulting U and V sets. While these rules, shown in Figure 4, are redundant, encoding the rules declaratively as part of the ASP encoding presented in this work improves the overall running times of our approach in practice.

Proposition 2 *Given a CG G over a node set N, starting with $U_x^C = V_x^C = Z_x^C = W_x^C = \emptyset$ for each node $x \in N$ and $C \subseteq N$, the following computations give the same U_x^C and V_x^C sets:*

- *Apply the rules 0–9 in Figure 2 until fixpoint.*
- *Apply the rules 0–9 in Figure 2 together with rules 12–17 in Figure 4 until fixpoint.*

Proof (sketch). Consider rule 12. According to the head of the rule, there exists a C-open route from a to x which does not contain the subroute $a_i \to a_{i+1} - \cdots - a_{i+k} = x$ with $k \geq 1$. Then, preceding this route with the edge $b \leftarrow a$ with $b \notin C$ results in a C-open route from b to x which does not contain the subroute $a_i \to a_{i+1} - \cdots - a_{i+k} = x$ with $k \geq 1$. Then, the body of the rule holds. The correctness of the rest of the rules can be proven in the same way. □

12. $x \in U_a^C, a \to b, b \notin C \Rightarrow x \in U_b^C$
13. $x \in V_a^C, a \to b, b \notin C \Rightarrow x \in V_b^C$
14. $x \in Z_a^C, a \to b, b \notin C \Rightarrow x \in Z_b^C$
15. $x \in U_a^C, y \in U_x^C \Rightarrow y \in U_a^C$
16. $x \in U_a^C, y \in V_x^C \Rightarrow y \in V_a^C$
17. $x \in U_a^C, y \in Z_x^C \Rightarrow y \in Z_a^C$

Figure 4: Additional sound rules

4 LEARNING CHAIN GRAPHS VIA ASP

In this section we detail our method of learning CGs using ASP. The approach is constraint-based and allows for the objective function to take additional domain knowledge into account. We start with a short informal account of ASP.

4.1 Answer Set Programming

Answer set programming (ASP) is a rule-based declarative constraint satisfaction paradigm that is well-suited for representing and solving various computationally hard problems (Gelfond and Lifschitz, 1988; Niemelä, 1999; Simons et al., 2002). ASP offers an expressive declarative modelling language in terms of first-order logical rules, allowing for intuitive and compact representations of NP-hard optimization tasks. When using ASP, the first task is to model the problem in terms of ASP rules (constraints) so that the set of solutions implicitly represented by the ASP rules corresponds to the solutions of the original problem. One or multiple solutions of the original problem can then be obtained by invoking an off-the-shelf ASP solver on the constraint declaration. The algorithms underlying the ASP solver Clingo (Gebser et al., 2011) that we use in this work are based on state-of-the-art Boolean satisfiability solving techniques (Biere et al., 2009). These techniques have during the last 10-15 years emerged as robust and efficient means of solving various hard search and optimization problems and even in cases surpassed specialized algorithms while at the same time offering great flexibility as general NP-procedures for declarative problem solving. As a self-contained explanation of ASP syntax and semantics would exceed the page limit, we only aim to give an intuitive reading of our ASP encoding for learning CGs.

4.2 Encoding the CG structure learning problem

Our exact ASP encoding of the CG structure learning problem is modular, consisting of three parts: (1) a set of constraints representing the space of CGs with a given set of nodes, including constraints ruling out semidirected cycles (Section 4.2.1); (2) a set of rules exactly encoding the Studený inference rules and thereby the separations and non-separations of a given CG (Section 4.2.2); (3) a set of soft constraints exactly representing a well-defined objective function used for finding the optimal CG structure of a problem (Section 4.2.3). Note that parts 2 and 3 are dependent on the input data. Specifically, part 2 represents the dependencies and independencies that are determined by the data. Additional information about the dependencies and independencies can also be added to part 2, such as the confidence of their correctness. This information can then be used by the objective function in part 3 as discussed in Section 4.2.3. Essentially, by calling an ASP solver with the whole ASP encoding, consisting of parts 1–3, the solver will perform an intelligent implicit search over the space of CGs (using part 1), and will output a CG that produces the best objective function score (based on part 3) by deriving the separations and non-separations in the CGs (using part 2). Furthermore, in Section 4.3 we discuss how additional domain knowledge can be incorporated.

4.2.1 Encoding CGs

We start with part 1, i.e., by describing an ASP program encoding of the space of CGs. This base encoding is presented in Figure 5.

In the encoding, the input predicate *node* represents the fact that x is a node. We use the predicates *edge* and *arc* to represent the undirected and directed edges, respectively, of the CG G, i.e., these predicates represent the actual CG described by a solution to the ASP program. More precisely, we have that $\mathit{edge}(X, Y)$ is true iff $x - y \in G$, and $\mathit{arc}(X, Y)$ is true iff $x \to y \in G$ Informally, the "guess" part of the program, consisting of the two first rules, encodes a non-deterministic guess of the edges in G, which means that the ASP solver will implicitly consider all possible graphs during search. The next three rules enforce the fact that, for any pair x, y of nodes in a CG G, there can be at most one type of an edge (undirected, directed, or neither) between them. The predicate $\mathit{ancestor}(X, Y)$ is true iff there is a semidirected route from a node x to node y, and is used to enforce that CGs cannot contain semidirected cycles. This constraint is enforced transitively by the last five rules in Figure 5.

```
%%% guess graph:
% directed edges
{ arc(X,Y) } :- node(X;Y), X!=Y,
                not edge(X,Y),
                not arc(Y,X).
% undirected edges
{ edge(X,Y) } :- node(X;Y), X!=Y,
                 not arc(X,Y),
                 not arc(Y,X).

%%% at most one edge between node pairs:
% symmetric undirected edge relation
edge(Y,X) :- edge(X,Y).
% disallow cases with both directed and
% undirected edges over a pair of nodes
:- edge(X,Y), arc(X,Y).
% disallow pairwise opposite arcs
:- arc(X,Y), arc(Y,X).

%%% disallow partially directed cycles
ancestor(X,X) :- node(X).
ancestor(X,Y) :- arc(X,Y).
ancestor(X,Y) :- edge(X,Y).
ancestor(X,Y) :- ancestor(X,Z),
                 ancestor(Z,Y).
:- ancestor(X,Y), arc(Y,X).
```

Figure 5: ASP encoding of chain graphs

4.2.2 Encoding Studeny's Rules

For inferring the separations that hold in a given CG G, we encode Studený's inference rules—or more precisely, the modified rule set consisting of the rules 0–4, 8–9, and 10–11 (recall Section 3)—in ASP. As shown in Figure 6, the ASP modelling language allows for very natural encoding of these rules. The predicates *inU*(X, A, C), *inV*(X, A, C), and *inZ*(X, A, C), respectively, are used for representing the facts that a node $x \in U_a^C$, $x \in V_a^C$, and $x \in Z_a^C$, respectively. The auxiliary predicate *in*(C, X), although its defining rule may seem somewhat complicated, simply represents the fact that node $x \in C \subseteq N$. As a technical detail, we use an index-representation for the conditioning sets and the nodes: the index of a given set $C \subseteq N = \{1..|N|\}$ is represented as a binary vector $b_{|N|} \dots b_2 b_1$, where $b_i = 1$ iff node $i \in C$. The sets $C \subseteq N$ are represented by the input predicate *set*. Finally, the predicate *derived_dep*(X, A, C) is defined to be true iff $x \not\perp\!\!\!\perp_G a | C$, where G is the graph represented by the predicates *edge* and *arc*, exactly following the conditions stated in Theorem 1.

```
% in(C,X): node X in set C
in(C,X) :- set(C), node(X),
           2**(X-1) & C != 0.
% rule 0
inU(X,X,C) :- set(C), node(X),
              not in(C,X).
% rule 1
inU(Y,A,C) :- inU(X,A,C), edge(X,Y),
              not in(C,Y), not in(C,A).
% rule 2
inU(Y,A,C) :- inU(X,A,C), arc(Y,X),
              not in(C,Y), not in(C,A).
% rule 3
inV(Y,A,C) :- inU(X,A,C), arc(X,Y),
              not in(C,Y), not in(C,A).
inV(Y,A,C) :- inV(X,A,C), arc(X,Y),
              not in(C,Y), not in(C,A).
% rule 4
inV(Y,A,C) :- inV(X,A,C), edge(X,Y),
              not in(C,Y), not in(C,A).
% rule 8
inZ(Y,A,C) :- inZ(X,A,C), edge(X,Y),
              not in(C,A).
% rule 9
inU(Y,A,C) :- inZ(X,A,C), arc(Y,X),
              not in(C,Y), not in(C,A).
% rule 10
inZ(Y,A,C) :- inU(X,A,C), arc(X,Y),
              not in(C,A), in(C,Y).
inZ(Y,A,C) :- inV(X,A,C), arc(X,Y),
              not in(C,A), in(C,Y).
% rule 11
inZ(Y,A,C) :- inV(X,A,C), edge(X,Y),
              not in(C,A), in(C,Y).

% Derived connections
derived_dep(X,Y,C) :- inU(Y,X,C), X != Y,
                      not in(C,Y), not in(C,X).
derived_dep(X,Y,C) :- inV(Y,X,C), X != Y,
                      not in(C,Y), not in(C,X).
```

Figure 6: ASP encoding of Studeny's rules

4.2.3 Optimization

We proceed by detailing how different objective functions can be encoded within our approach. As noted earlier, the goal of an objective function is to assign a score to every CG G, and the set of optimal CGs is defined as those CGs which minimizes the objective function value. We consider three different objective functions that characterize different types of optimality conditions for CGs. Following Hyttinen et al. (2014)—who focused on causal structure discovery—each of the objective functions is (in some cases partially) based on how well the separations and non-separations of the given CG G corresponds to the independencies and dependencies determined from the data. In the following, we denote by $\mathrm{CG}(N)$ the set of CGs over the set of nodes N, and by I the (complete) set of independencies and dependencies given as input (determined before-hand from the data).

The first two objective functions follow the general idea of minimizing a sum of costs over incorrect separations and non-separations in G wrt the independences and dependences determined by the data. As a generalization, we associate for each (in)dependence statement determined by the data a cost (weight) $w(x \perp\!\!\!\perp y | C)$ and $w(x \not\perp\!\!\!\perp y | C)$, which is incurred on the solution G iff the corresponding separation statement does not hold in G. This allows for formalizing a general weighted objective function

$$\min_{G \in \mathrm{CG}(N)} \quad \sum_{(x \not\perp\!\!\!\perp y | C) \in I} w(x \not\perp\!\!\!\perp y | C) \cdot T[x \perp\!\!\!\perp y | C] + \sum_{(x \perp\!\!\!\perp y | C) \in I} w(x \perp\!\!\!\perp y | C) \cdot T[x \not\perp\!\!\!\perp y | C], \quad (1)$$

where $T[c]$ is an indicator for the condition c being true for graph G.

By setting the weights w appropriately, this gives various interesting special cases. For example, to represent independence optimal CGs as the optimal solutions, let $w(x \not\perp\!\!\!\perp y | C) = \infty$ (i.e., all dependencies have to be represented in any CG G) and $w(x \perp\!\!\!\perp y | C) = 1$ (i.e., each independence relation not represented by G adds a unit cost to G).

On the level of our ASP encodings of the objective functions considered, we use the input predicates *indep*(X, Y, C, W) and *dep*(X, Y, C, W), to represent $w(x \perp\!\!\!\perp y | C)$ and $w(x \not\perp\!\!\!\perp y | C)$, respectively. Using these predicates, the general weighted objective function is expressed as the ASP statements shown in Figure 7.

In addition to allowing for associating the same (unit) weight to all (in)dependence statements, the general weighted objective function also allows for using more elaborate weighting schemes where each (in)dependence statement have a different weight. In contrast, most

```
% Minimize the sum of the weights of
% unsatisfied (in)dependence constraints
:~indep(X,Y,C,W), derived_dep(X,Y,C).
                                [W,X,Y,C]
:~dep(X,Y,C,W), not derived_dep(X,Y,C).
                                [W,X,Y,C]
```

Figure 7: Encoding the general weighted objective

```
% satisfy all dependence constraints
:- dep(X,Y,C,_), not derived_dep(X,Y,C).

% minimize undirected and directed edges
:~ edge(X,Y), X<Y. [1,X,Y]
:~ arc(X,Y). [1,X,Y]
```

Figure 8: Encoding the minimize-edges objective

constraint-based structure learning methods only allow for binary information about an independence statement, which means that possible additional information about the independence statements, such as the level of confidence in them, is lost. In fact, as suggested by Hyttinen et al. (2014), by developing weighting schemes, one can bring the constraint-based approaches closer to score-based methods which have traditionally used local scores as weights within the implemented objective functions. For example, a weighting scheme that was proposed and shown to produce good solutions by Hyttinen et al. (2014) uses the log of the Bayesian probability of (in)dependence statements (Margaritis and Bromberg, 2009) as weights.

As an alternative to expressing optimality in terms of weighting incorrectly represented (in)dependencies, we also consider a novel objective function that aims at minimizing the number of edges in the CG while at the same time representing all conditional dependencies in the data. The underlying idea of the objective function is to minimize the model complexity and although independence optimal solutions can no longer be guaranteed wrt the independence model of the data, any solution will still be inclusion optimal. Formally, this gives the alternative objective function

$$\min_{G \in \mathrm{CG(N)}} \quad |\{(x,y) \,:\, x < y,\ x - y \in G\}| + |\{(x,y) \,:\, x \to y \in G\}| + \sum_{(x \not\perp\!\!\!\perp y|C) \in I} w(x \not\perp\!\!\!\perp y|C) \cdot T[x \perp\!\!\!\perp_G y|C], \quad (2)$$

where $w(x \not\perp\!\!\!\perp y|C) = \infty$. This objective function is encoded in ASP as shown in Figure 8, which in itself demonstrates the versatility of our declarative approach.

4.3 Imposing Additional Constraints

One of the major strengths with our CG learning approach is the possibility to incorporate further constraints to guide the search for optimal solutions, in order to e.g. speed up the search for solutionor to focus the search on specific solutions of interest. In terms of domain knowledge, if we know that variable x is the direct cause of variable y in the system we are modelling, we can simply add the constraint $arc(x, y)$ as input to the ASP solver. In terms of including redundant constraints for speeding up search, one can e.g. encode the set of redundant rules for how the sets U_x^C, V_x^C and Z_x^C can be computed as discussed in Section 3.2. An encoding of the rules are shown in Figure 9. In fact, we observed that the solver running times decreased by approximately 50% after adding these rules to the ASP encoding.

In terms of focusing search specific solutions of interest, we now describe an ASP encoding for restricting the search to only consider so-called *largest chain graphs* (Frydenberg, 1990; Volf and Studený, 1999). Largest CGs are representatives of Markov equivalence classes of CGs. Given a Markov equivalence class, the largest CG in the class is the CG which includes a maximal number of undirected edges. More formally, largest chain graphs are defined as follows.

A *complex* in a CG G is a path $v_1, \ldots, v_k$ in G with $k \geq 3$, such that $v_1 \to v_2$, $v_i - v_{i+1}$ for $i = 2, \ldots, k-2$, and $v_{k-1} \leftarrow v_k$, and there are no other edges between the nodes $v_1, \ldots, v_k$ in G. A *complex edge* is a directed edge that belongs to a complex. A directed edge $u \to v$ *covers* a directed edge $x \to y$ in a CG G u is an ancestor of x and y is an ancestor of v in G. A directed edge $u \to v$ is *protected* in G if it covers a complex edge. Note that every complex edge is protected. Now, a CG G is the largest CG in a Markov equivalence class iff every directed edge of G is protected.

An ASP encoding of the largest CG constraint is presented in Figure 10. Here, the predicate *complex*(X, Y) indicates that $x \to y$ is a complex edge. The auxiliary predicate *almost_undirected_path*(X, Y, Z) indicates the existence of a path $x \to y - \cdots - z$ such that there are no other edges between the nodes in the path. This predicate

```
inU(Z,Y,C) :- inU(Z,X,C), arc(X,Y),
   not in(C,X), not in(C,Y), not in(C,Z).
inV(Z,Y,C) :- inV(Z,X,C), arc(X,Y),
   not in(C,X), not in(C,Y), not in(C,Z).
inZ(Z,Y,C) :- inZ(Z,X,C), arc(X,Y),
                 not in(C,X), not in(C,Y).

inU(Y,A,C) :- inU(X,A,C), inU(Y,X,C).
inV(Y,A,C) :- inU(X,A,C), inV(Y,X,C).
inZ(Y,A,C) :- inU(X,A,C), inZ(Y,X,C).
```

Figure 9: Encoding redundant domain knowledge

```
% derive complex arcs X->Y - ... <-Z
complex(X,Y) :- arc(X,Y), arc(Z,Y), X!=Z,
                not edge(X,Z),
                not arc(X,Z), not arc(Z,X).
complex(X,Y) :- almost_undirected_path(X,Y,V),
                almost_undirected_path(Z,V,Y),
                X!=Z, not edge(X,Z),
                not arc(X,Z), not arc(Z,X).
almost_undirected_path(X,Y,Z) :- arc(X,Y),
                edge(Y,Z), not arc(X,Z).
almost_undirected_path(X,Y,Z) :-
                almost_undirected_path(X,Y,V),
                edge(V,Z), not arc(X,Z).

% arc U->V covers arc X->Y
covers(U,V,X,Y) :- ancestor(U,X),
                   ancestor(Y,V),
                   arc(X,Y), arc(U,V).

% all arcs must be protected
1 {covers(U,V,X,Y) : complex(X,Y)} :-arc(U,V).
```

Figure 10: ASP encoding of the largest CG constraint

is used for deriving the predicate *complex*. The predicate *covers*(U, V, X, Y) encodes that $u \to v$ covers the edge $x \to y$. The final cardinality constraint states that every directed edge $u \to v$ must cover at least one complex edge $x \to y$, i.e., $u \to v$ has to be protected.

5 EMPIRICAL EVALUATION

We report here on an empirical evaluation comparing our declarative ASP-based approach with existing CG structure learning algorithms in terms of accuracy of the produced solutions. For solving the ASP encodings, we used the state-of-the-art ASP solver Clingo version 4.4.0. The CG learning algorithms we compare to are LCD (Ma et al., 2008) and CKES (Peña et al., 2014), for which we obtained implementations from the respective authors. We excluded the PC-like algorithm (Studený, 1997) from the comparison since Ma et al. (2008) have shown that it is outperformed by LCD.

For the ASP approach, we illustrate its diversibility by applying three different objective functions.

(i) ASP-Indep, which implements the general weighted optimization function with $w(x \not\perp\!\!\!\perp y|C) = \infty$ and $w(x \perp\!\!\!\perp y|C) = 1$ (cf. Section 4.2.3, Equation 1);

(ii) ASP-Weight, which implements the general weighted optimization function using a probability-based weighting scheme from (Hyttinen et al., 2014, Section 4.3), following a Bayesian paradigm to assign probabilities to (in)dependence statements, using code from the authors;

(iii) ASP-Edge, which implements the minimize edges optimization function, (cf. Section 4.2.3, Equation 2).

Since the LCD algorithm assumes faithfulness, to some CG, from the probability distribution it is trying to learn, and does not work properly otherwise, we enforced this assumption for the experiments. However, we want to emphasize that our approach works also without such assumptions. Hence, due to this strong fairness towards LCD and CKES, the empirical results may to some extent present overly positive results for LCD and CKES.

Assuming a faithful underlying probability distribution, we applied the following process. First, a set of CGs G were generated with corresponding probability distributions. These probability distributions were then sampled into datasets, each determining a set of independence and dependence statements, forming the inputs to the different algorithms. We then evaluated the learning results, i.e., the CGs output as solutions by the algorithms, by comparing them to the original CG G.

For the experiments, we generated 100 "original" CGs over $N = 7$ nodes, with an average node-degree of 2, as well as corresponding Gaussian probability distributions. For each distribution, we obtained 500 samples. We used the simplified Wilks independence test (Wilks, 1938) for LCD, CKES, ASP-Indep and ASP-Edge. For the ASP-Weight variant, we used the more advanced, non-binary independence test following (Hyttinen et al., 2014). Taking into account that CKES typically ends up in different local optima (Peña et al., 2014), we ran the CKES algorithm three times on each input, and report the best found solution—wrt independence optimality to the independence model determined by the data—as the final solution produced by CKES.

To evaluate the accuracy of the learnt CGs we compared their represented independence model with the independence model of the original CG G by calculating the true positive rate (TPR) and false positive rate (FPR) of learnt dependencies. A higher TPR means that more dependencies are correctly identified by the algorithm, while a lower FPR means that more independencies are correctly identified.

The accuracy of the results is plotted in the ROC space in Figure 11.[1] We observe that LCD does not achieve as high TPR as the other algorithms, but does obtain relatively low FPR. This is in line with earlier results (Ma et al., 2008; Peña et al., 2014) that show that LCD includes edges relatively cautiously. CKES, in turn, achieves relatively high TPR, but with higher FPR, which is in line with previous evidence suggesting that CKES can have problems identifying the correct independencies in a probabil-

[1]Note that the points are not statistical test results obtained independently of each other. Rather, they are learning results of algorithms for which the dependencies and independencies logically constrain each other. As a result, some of the algorithms are not able to produce results with low FPRs at all, making the curves non-concave.

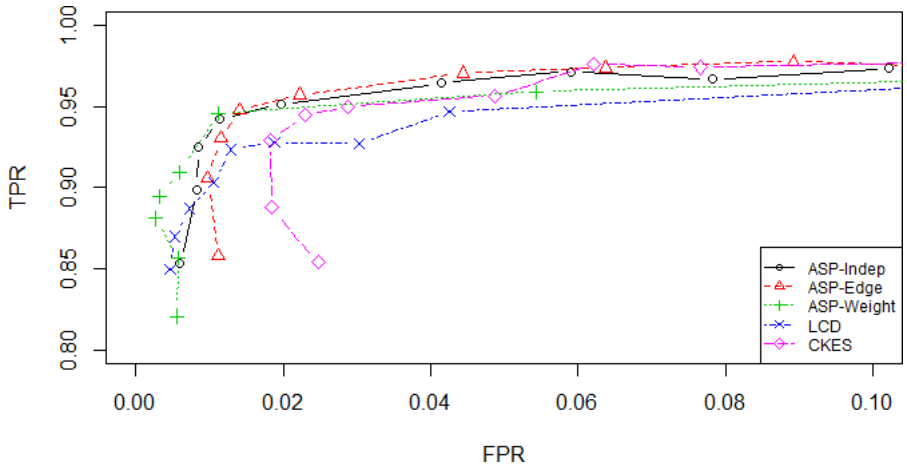


Figure 11: Comparison of solution accuracy

ity distribution and hence is relatively generous in including edges (Peña et al., 2014). In contrast, relative to LCD and CKES, we observe that the ASP-based approach under the different objective functions perform well both in the low FPR range (similarly or better than LCD which in turn dominates CKES in the range) and the higher FPR range (similarly or better than CKES which is in turn dominates LCD in the range). Thus the overall best-performing approaches are the ASP encoding variants. Among the ASP encoding variants, we observe that the ASP-Weight variant shows best performance in the low FPR range < 0.01. The variants ASP-Weight and ASP-Indep, using instantiations of the general weighted objective function, fare better than ASP-Edge that optimally minimizes the number of edges in the output CG.

As for scalability of our approach, the average and median running times over 100 inputs in terms of the number of variables are shown in Table 1, using a single core of an Intel i5 processor running at 3.4 GHz. For a fair comparison, we note that CKES and LCD do exhibit much faster running times, e.g., 1-10 seconds per run at $N = 7$. Note that the median ASP running times are much lower then the average running times. We observed that relatively scarce outliers have a big negative influence on the averages. The running times include both the time it takes to reach an optimal solution as well as the additional time it takes for the solver to *prove* that the found solution is optimal. In fact, typically proving optimality consumes the majority of the running time of the solver. Moreover, during the search, the solver actually reports increasingly good solutions, constituting an anytime learning algorithm.

Table 1: ASP running times under different objectives

N	**Median / Average (sec)**		
	ASP-Indep	**ASP-Edge**	**ASP-Weight**
3	0.02 / 0.02	0.02 / 0.02	0.02 / 0.02
4	0.03 / 0.03	0.04 / 0.04	0.03 / 0.03
5	0.11 / 0.15	0.33 / 0.35	0.24 / 0.46
6	0.93 / 2.06	7.84 / 10.10	5.00 / 270.35
7	21.35 / 243.10	366.86 / 489.71	60.70 / 2471.81

6 CONCLUSIONS

In this article we presented a first exact approach to chain graph structure learning. In the approach the computationally hard problem is cast as a constraint optimization problem using the declarative programming of answer set programming. In contrast to previously proposed CG learning algorithms, our approach enables finding provably optimal solutions to the learning problem without making assumptions on the data. It is at the same time more general since it enables the use of various types of objective functions for optimization and specifying domain knowledge in terms of hard constraints. Via an empirical evaluation we also showed that the approach exhibits better overall accuracy than any single previously proposed CG structure learning algorithm. A topic for further work is to study ways of scaling up the approach to higher numbers of variables while still maintaining optimality guarantees. It would also be interesting to extend the approach to accommodate the alternative multivariate regression (Cox and Wermuth, 1993, 1996) and Andersson-Madigan-Perlman (Andersson et al., 2001; Levitz et al., 2001) CG interpretations.

Acknowledgements This work was supported in part by the Swedish Research Council (2010-4808), Academy of Finland (grants 251170 COIN Centre of Excellence in Computational Inference Research, 276412, and 284591), and Research Funds of the University of Helsinki.

References

Andersson, S., Madigan, D., and Perlman, M. (2001). An alternative Markov property for chain graphs. *Scandinavian Journal of Statistics*, 28:33–85.

Berg, J., Järvisalo, M., and Malone, B. (2014). Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In *Proc. AISTATS*, volume 33 of *JMLR Proceedings*, pages 86–95. JMLR.org.

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

Corander, J., Janhunen, T., Rintanen, J., Nyman, H. J., and Pensar, J. (2013). Learning chordal markov networks by constraint satisfaction. In *Proc. NIPS*, pages 1349–1357.

Cox, D. and Wermuth, N. (1993). Linear dependencies represented by chain graphs. *Statistical Science*, 8:204–218.

Cox, D. and Wermuth, N. (1996). *Multivariate Dependencies: Models, Analysis and Interpretation*. Chapman and Hall.

Cussens, J. and Bartlett, M. (2013). Advances in Bayesian network learning using integer programming. In *Proc. UAI*, pages 182–191. AUAI Press.

Frydenberg, M. (1990). The chain graph Markov property. *Scandinavian Journal of Statistics*, 17:333–353.

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., and Schneider, M. T. (2011). Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124.

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In *Proc. ICLP*, pages 1070–1080.

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

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

Lauritzen, S. and Wermuth, N. (1989). Graphical models for association between variables, some of which are qualitative and some quantitative. *The Annals of Statistics*, 17:31–57.

Levitz, M., Perlman, M., and Madigan, D. (2001). Separation and completeness properties for AMP chain graph Markov models. *The Annals of Statistics*, 29:1751–1784.

Ma, Z., Xie, X., and Geng, Z. (2008). Structural learning of chain graphs via decomposition. *Journal of Machine Learning Research*, 9:2847–2880.

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

Nielsen, J., Kočka, T., and Peña, J. (2003). On local optima in learning Bayesian networks. In *Proc. UAI*, pages 435–442. AUAI Press.

Niemelä, I. (1999). Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273.

Parviainen, P., Farahani, H. S., and Lagergren, J. (2014). Learning bounded tree-width Bayesian networks using integer linear programming. In *Proc. AISTATS*, volume 33 of *JMLR Proceedings*, pages 751–759. JMLR.org.

Peña, J. M., Sonntag, D., and Nielsen, J. (2014). An inclusion optimal algorithm for chain graph structure learning. In *Proc. AISTATS*, volume 33 of *JMLR Proceedings*, pages 778–786. JMLR.org.

Simons, P., Niemelä, I., and Soininen, T. (2002). Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234.

Sonntag, D., Peña, J., and Gómez-Olmedo, M. (2015). Approximate counting of graphical models via MCMC revisited. *International Journal of Intelligent Systems*, 30:384–420.

Studený, M. (1997). On recovery algorithms for chain graphs. *International Journal of Approximate Reasoning*, 17:265–293.

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

Studený, M. (2005). *Probabilistic Conditional Independence Structures*. Springer.

Volf, M. and Studený, M. (1999). A graphical characterization of the largest chain graphs. *Int. J. Approx. Reasoning*, 20(3):209–236.

Wilks, S. (1938). The large-sample distributiom of the likelihood ratio for testing composite hypotheses. *The Annals of Mathematical Statistics*, 20:595–601.