Department of Computer Science
Series of Publications A
Report A-2021-7

Optimization Algorithms for Learning Graphical Model Structures

Kari Rantanen

Doctoral dissertation, to be presented for public examination with the permission of the Faculty of Science of the University of Helsinki on December 8, 2021 at 15 o'clock. The defence is open for audience through remote access.

University of Helsinki
Finland

Supervisors

Matti Järvisalo, University of Helsinki, Finland
Antti Hyttinen, University of Helsinki, Finland

Pre-examiners

Cassio de Campos, Eindhoven University of Technology, The Netherlands
Pekka Parviainen, University of Bergen, Norway

Opponent

Peter van Beek, University of Waterloo, Canada

Custos

Matti Järvisalo, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911

ISSN 1238-8645
ISBN 978-951-51-7749-0 (paperback)
ISBN 978-951-51-7750-6 (PDF)
Helsinki 2021
Unigrafia

Optimization Algorithms for Learning Graphical Model Structures

Kari Rantanen

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
kari.rantanen@helsinki.fi

PhD Thesis, Series of Publications A, Report A-2021-7
Helsinki, December 2021, 68+76 pages
ISSN 1238-8645
ISBN 978-951-51-7749-0 (paperback)
ISBN 978-951-51-7750-6 (PDF)

Abstract

Graphical models are a versatile machine learning framework enabling efficient and intuitive representations of probability distributions. They can be used for performing various data analysis tasks that would not be feasible otherwise. This is made possible by constructing a graph structure which encodes the underlying dependence structure of the probability distribution. To that end, the field of structure learning develops specialized algorithms which can learn a structure that describes given data well.

This thesis presents advances in structure learning for three different graphical model classes: chordal Markov networks, maximal ancestral graphs, and causal graphs with cycles and latent confounders. Learning structures for these model classes has turned out to be a very challenging task, with the few existing exact methods scaling to much fewer number of random variables compared to more extensively developed methods for Bayesian networks.

Chordal Markov networks are a central class of undirected graphical models. Being equivalent to so-called decomposable models, they are essentially a special case of Bayesian networks. This thesis presents an exact branch-and-bound algorithm and an in-exact stochastic local search for learning chordal Markov network structures. Empirically we show that the branch and bound is at times able to learn provably optimal structures with higher number of variables than competing methods, whereas the local search scales considerably further.

Maximal ancestral graphs are a generalization of Bayesian networks which allow for representing the influence of unobserved variables. This thesis introduces the first exact search algorithm for learning maximal ancestral graphs and a framework for generating and pruning local scores for the search. Empirically we show that the proposed exact search is able to learn higher-quality structures than existing in-exact methods.

Finally, we introduce an exact branch-and-bound algorithm for learning causal graphs in the presence of cycles and latent confounders. Our empirical results show that the presented algorithm is able to learn optimal structures considerably faster than a recent exact method for the problem. We also extend our branch and bound to support interventional data and σ-separation, and show empirically that the algorithm can handle higher number of experimental datasets than the only competing method supporting σ-separation.

Computing Reviews (2012) Categories and Subject Descriptors:

Theory of computation → Design and analysis of algorithms → Algorithm design techniques → Branch-and-bound and Dynamic programming
Theory of computation → Design and analysis of algorithms → Mathematical optimization → Discrete optimization → Optimization with randomized search heuristics → Randomized local search
Discrete mathematics → Combinatorics → Combinatorial optimization
Discrete mathematics → Graph theory → Graph algorithms
Probability and statistics → Probabilistic representations → Bayesian networks, Markov networks and Causal networks

General Terms:
algorithms, design, theory, experimentation

Additional Key Words and Phrases:
structure learning, Bayesian networks, causal discovery, chordal Markov networks, decomposable models, maximal ancestral graphs, stochastic local search, branch and bound, dynamic programming

Acknowledgements

This thesis is an end result of the research I conducted in the Doctoral Programme in Computer Science (DoCS) at the University of Helsinki. I am glad for having had the chance to work in the Constraint Reasoning and Optimization (CoReO) research group with many great people who provided me a warm work environment.

Especially I wish to thank my supervisors Matti Järvisalo and Antti Hyttinen for guiding my doctoral studies over the years and for collaborating with me on the research presented in this dissertation. I am also grateful to Prof. Järvisalo for providing funding for my doctoral studies.

I would like to thank Peter van Beek for agreeing to act as the opponent of this dissertation. I also appreciate the valuable feedback Cassio de Campos and Pekka Parviainen provided as the pre-examiners of this thesis. I am also deeply grateful to Research Coordinator Pirjo Moen for all the help she has given me throughout my doctoral studies.

Finally, I wish to thank the Finnish Grid and Cloud Infrastructure (FGCI) for supporting this project with computational and data storage resources.

Helsinki, November 2021
Kari Rantanen

Contents

Chapter 1

Introduction

Probability distributions are used extensively in science, medicine and AI to make sense of the uncertainty surrounding objects and events [41, 52, 61]. As an example, a probability distribution can help us predict the disease of a patient given their symptoms. In this case the diseases and symptoms would be the *random variables* of the distribution.

A probability table offers one way to represent a discrete probability distribution [41]. The table could be constructed by gathering empirical data of a studied phenomenon, for example in the form of patient records, and then using these statistics to compute joint probabilities between the variables. However, constructing and storing the complete probability table quickly becomes infeasible for distributions over many variables, simply because the number of probabilities grows exponentially with respect to the number of variables [60].

For this reason having compact representations for probability distributions is desirable. One way to achieve this is to take advantage of the underlying structure of the distribution, namely the *statistical dependencies* between the variables [16, 41]. Indeed, if a set of variables are independent of each other, then there is no point in storing the joint probabilities of those variables. This observation gives rise to *(probabilistic) graphical models* which encode statistical dependencies via a graph structure [60]. The nodes of the graph correspond to the random variables of the distribution and the edge relation encodes statistical dependencies between the variables. A graphical model thus allows for not explicitly storing probabilities between non-adjacent variables in the graph, drastically reducing the amount of parameters needed to represent a probability distribution.

We proceed by providing a brief overview of the most relevant graphical model classes for this thesis. In particular, we focus on chordal Markov networks, max-

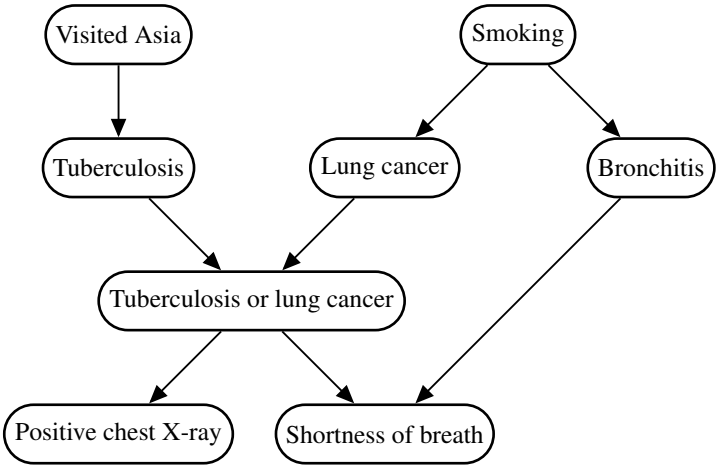


Figure 1.1: A Bayesian network structure [43].

imal ancestral graphs, and causal graphs with cycles and latent variables. Each of these model classes is either a special case or a generalization of Bayesian networks, which will be the starting point of the following discussion.

Bayesian networks (BNs) are one of the most well-known classes of graphical models [41, 60, 61]. BNs encode statistical dependencies in the form of *directed acyclic graph* (DAG), meaning that all the edges of the structure are directed and there are no directed cycles. A classic example of a Bayesian network structure by Lauritzen and Spiegelhalter [43] is illustrated in Figure 1.1. Visiting Asia can increase the risk of tuberculosis while smoking can cause lung cancer and bronchitis, both of which, like tuberculosis, can in turn cause shortness of breath. Yet shortness of breath or a positive chest X-ray are not enough in themselves to discriminate between lung cancer and tuberculosis. In this case not only do the edges denote dependencies between the variables but also the underlying causal relations. However, in general, the edge directions of a Bayesian network do not have to have such causal interpretations [41].

Chordal Markov networks are a central class of undirected graphical models [18, 44, 47, 95]. The term "chordal" refers to the fact that each cycle with four or more nodes in the graph has an edge (called a chord) connecting two non-consecutive nodes of the cycle. Interestingly, when the edges of chordal Markov networks are directed in a certain way, they are equivalent to decomposable models, a special case of Bayesian networks [18, 41]. Figure 1.2 illustrates an example chordal Markov network structure by Edwards [24]. The structure is based on data containing the examination marks of 88 students on five different topics. The graph shows how success in the different topics is linked to each other, revealing, for

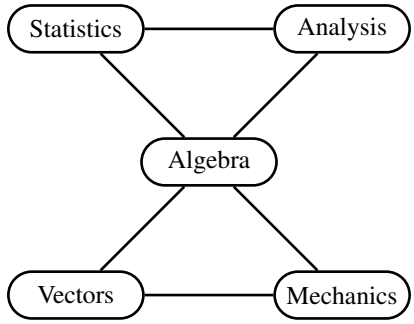


Figure 1.2: A chordal Markov network structure [24].

example, that the students' examination marks in statistics were conditionally independent of their marks in mechanics given their marks in algebra. Due to the undirected nature of chordal Markov networks, they are suited for modelling situations like this where there is no clearly defined causal direction between the variables.

Maximal ancestral graphs (MAGs) are a generalization of Bayesian networks which allow for representing latent confounding via bidirected edges [92]. Latent confounding means that some unobserved variable is a common cause of two nodes in the graph. MAG structures are restricted by certain maximality and ancestrality requirements which constrain, for example, that they cannot contain directed cycles. An example MAG is illustrated in Figure 1.3. It is derived from Figure 1.1 by assuming that some of the variables (namely, "Smoking", "Visited Asia", "Tuberculosis" and "Tuberculosis or lung cancer") are unobserved. The result is a graph over the remaining nodes in which, for example, the bidirected edge between "Lung cancer" and "Bronchitis" encodes that they have an unobserved common cause.

Causal graphs considered in this thesis are a further generalization of Bayesian networks [35]. Like MAGs, causal graphs can encode latent confounding via bidirected edges [61, 80]. In addition, they are able to contain directed cycles in order to represent cyclic causal structures [66, 68, 79]. For this reason a pair of nodes in a causal graph can be adjacent via multiple (up to three) edges.

Figure 1.4 shows visually the relative structural complexity between the beforementioned model classes. As already motivated, chordal Markov networks correspond to a certain subclass of Bayesian networks whereas maximal ancestral graphs extend Bayesian networks by allowing the presence of bidirected edges. Causal graphs with cycles and latent confounders are the most general model class considered in this thesis.

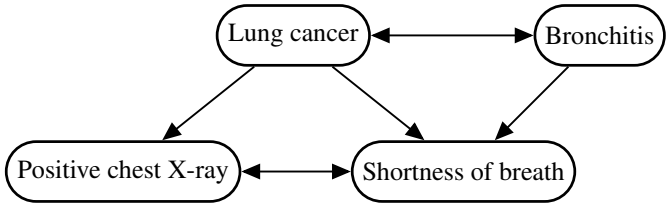


Figure 1.3: A maximal ancestral graph, derived from Figure 1.1.

Different classes of graphical models are useful for different purposes. If the chosen model class does not suit particular data, we may run into overfitting and underfitting issues [52]. Being unable to capture the true structure can cause the model to yield inaccurate predictions as well as to mislead us about the relationship between the variables. For these reasons it is important that there are methods readily available for learning structures for different kinds of graphical model classes.

Determining a well-fitting graph structure for given data is a challenging problem. In this thesis we focus on solving this problem with *structure learning*; that is, developing algorithms to find such structures based on given data [52]. This provides similar benefits as machine learning in general: we are able to process much larger amounts of data than what is possible for humans and the learning is automatized. In addition, machine learning can potentially reveal aspects of the data that are left unnoticed by domain experts.

Most structure learning algorithms can be roughly categorized into constraint-based and score-based approaches depending on how they find the structure, although there are also hybrid methods [27, 33, 35] combining both techniques. *Constraint-based* algorithms [4, 10, 11, 12, 64, 100] perform statistical tests on the data to identify conditional independence constraints. These constraints can then be used to determine which edges should be present in the graph structure. In contrast, *score-based* algorithms [8, 40, 45, 78, 85, 92, 93] assign a numerical value ("score" or "weight") to each graph and use this to determine which structures are the best fit for the data. This allows for formulating structure learning in terms of combinatorial optimization, either minimizing the weight or maximizing some well-justified score of the structure.

Structure learning algorithms can also be categorized into *exact* [38, 40, 58, 59, 78, 93] and *in-exact* [8, 10, 12, 29, 46, 64, 74] (or approximate) approaches. Given enough time, exact approaches are guaranteed to find an optimal structure eventually, whereas approximate methods do not provide such guarantees for the quality of the found solutions.

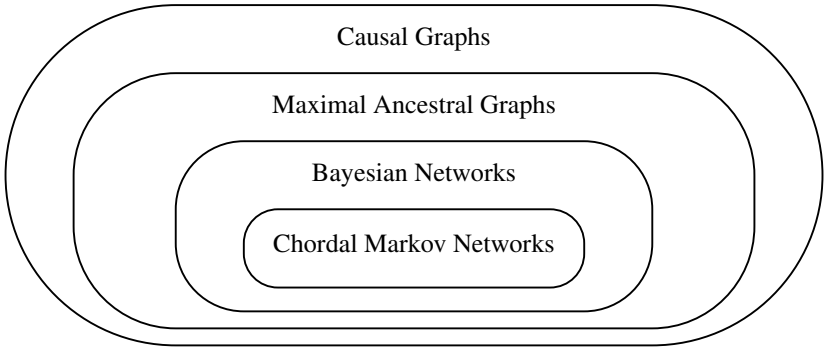


Figure 1.4: A diagram visualizing the relative structural complexity between four graphical model classes.

Most methods developed for *chordal Markov network structure learning* (CMSL) have been in-exact [5, 30, 49, 53, 63]. This is mostly due to how computationally challenging the problem is: beyond learning optimal tree-shaped networks (e.g. by using the polynomial-time Chow-Liu algorithm [9]), finding a maximum likelihood chordal Markov network with bounded treewidth is known to be NP-hard [82, 83]. Early algorithms for the problem considered greedy procedures which either added edges to a network or removed edges from a network while locally maximizing a given scoring function [21]. Since then more sophisticated techniques, such as graph cuts [77], convex relaxations [42] and generalizing the Chow-Liu algorithm for non-trees [2], have been proposed. More recently the development of exact approaches for CMSL has received noticeable attention. The first non-trivial exact method was presented by Corander *et al.* [14] and Janhunen *et al.* [38], characterizing chordal graphs as so-called junction trees [18] and using declarative solvers to find the optimal structure. A junction tree characterization was also adopted by Kangas *et al.* [39, 40] who developed a dynamic programming algorithm called Junctor for the problem. Studený and Cussens [85] proposed an integer programming based method for learning optimal chordal Markov networks by extending an earlier cutting plane approach GOBNILP [3] originally aimed for learning Bayesian networks.

Unlike in the case of CMSL, no exact methods have been developed for *maximal ancestral graph structure learning (MAGSL)*. There are, however, a number of in-exact approaches capable of learning MAGs or their equivalence classes. One of the earliest such methods is FCI by Spirtes *et al.* [80], a generalization of PC [80] which in turn builds upon earlier work by Verma and Pearl [94]. The algorithm is constraint-based and shown to be asymptotically correct in the presence of latent confounders. FCI has since then inspired a few variants [12, 13, 64].

Notably, Ogarrio *et al.* [56] proposed a hybdrid method called GFCI which combines FCI with a score-based algorithm called GES by Chickering [8]. Recently, Tsirlis *et al.* [92] presented a greedy hill climbing approach for MAGSL called M3HC, building upon earlier developments for GSMAG and MMHC [89,91]. Albeit score-based, the algorithm integrates constraint-based reasoning for deducing which pairs of nodes should not be adjacent.

One of the earliest methods proposed for learning cyclic causal structures is the in-exact CCD algorithm by Richardson [66]. Hyttinen *et al.* [33] developed an answer set programming (ASP) [50, 55] encoding for the task, building upon ideas from earlier declarative methods [34, 90]. It represents the first exact method for learning causal graphs with cycles and latent confounders that optimally satisfy given weighted (in)dependence constraints. Since then Forré and Mooij [27] have adapted the ASP encoding to learn optimal causal graphs in the presence of non-linear cyclic relations. A recent constraint-based exact approach for learning causal graphs with cycles and latent confounders is Dseptor by Hyttinen *et al.* [35]. It adapts a hybrid maximum satisfiability (MaxSAT) solver [72] to solve an implicit hitting set problem [17, 51, 73] over unsatisfiable cores for learning optimal causal graphs.

This thesis presents exact and in-exact structure learning approaches employing constraint-based and score-based techniques. The most important algorithmic techniques considered in this thesis are dynamic programming, branch and bound and stochastic local search.

Dynamic programming [15, 39, 78] solves a problem exactly by implicitly considering every possible solution candidate. This algorithmic paradigm requires that the problem can be recursively divided into increasingly small sub-problems which together provide a solution to the original problem. A technique called *memoization* caches solutions to already-solved sub-problems and prevents the dynamic programming from solving same sub-problems repeatedly.

Branch and bound [57,86,87,93] is either a depth-first or breadth-first backtracking algorithm. It usually defines a class of *partial solutions* which are gradually extended towards feasible solution candidates. "Branching" refers to the way partial solutions are extended during the search whereas "bounding" is a technique used for determining when the current partial solution cannot lead to a good solution candidate, often considerably reducing the search space.

(Stochastic) local search [31,46,96] is an in-exact approach to solving optimization problems. It explores the search space by moving from one solution candidate to another and keeps track of the best solution (by some metric) encountered so far. Moves through the search space are dictated by the notion of *neighbor* which refers to solutions which are close to each other. The search can combine algo-

rithmic greediness (i.e., making locally optimal choices) with randomized steps to prevent the search from getting stuck in a local optima.

This thesis contributes to the field of structure learning by presenting advances for learning chordal Markov network structures, maximal ancestral graphs and causal graphs. We propose optimization algorithms for solving these problems, implement them in the form of computer programs and evaluate the implementations empirically against the relevant competing algorithms for the respective problems. We also present some theoretical results relating to graphical model structures. The implementations of the algorithms are available online in open source.

1.1 Original Publications

The following five publications form the core of this thesis. We will refer to these as Papers I-V. The publications are reprinted at the end of this thesis.

I. Kari Rantanen, Antti Hyttinen, and Matti Järvisalo. Learning Chordal Markov Networks via Branch and Bound. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 1845–1855. Curran Associates, Inc., 2017.

II. Kari Rantanen, Antti Hyttinen, and Matti Järvisalo. Learning Chordal Markov Networks via Stochastic Local Search. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 2632–2639. IOS Press, 2020.

III. Kari Rantanen, Antti Hyttinen, and Matti Järvisalo. Maximal Ancestral Graph Structure Learning via Exact Search. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021)*, 2021. (To appear)

IV. Kari Rantanen, Antti Hyttinen, and Matti Järvisalo. Discovering causal graphs with cycles and latent confounders: An exact branch-and-bound approach. *International Journal on Approximate Reasoning*, 117:29–49, 2020.

V. Kari Rantanen, Antti Hyttinen, and Matti Järvisalo. Learning Optimal Cyclic Causal Graphs from Interventional Data. In *Proceedings of the 10th International Conference on Probabilistic Graphical Models (PGM 2020)*, volume 138 of *Proceedings of Machine Learning Research*, pages 365–376. PMLR, 2020.

Paper I introduces a branch-and-bound algorithm for learning optimal chordal Markov network structures. The search operates over decomposable DAGs and uses Bayesian network structures as upper bounds. The algorithm also proposes a way to break symmetries in the space of decomposable DAGs as well as a method for pruning local scores on the fly. The approach is able to solve up to 20-variable problem instances within 24 hours which was not possible for earlier algorithms without enforcing structural restrictions.

Paper II develops a stochastic local search algorithm for learning chordal Markov network structures. It uses on-the-fly score computation to avoid computing unnecessary local scores. Our empirical results show that the approach often finds optimal chordal Markov network structures and does so considerably faster than the exact state of the art. It can also scale towards hundreds of variables without enforcing structural restrictions.

Paper III focuses on learning maximal ancestral graph structures. We introduce the first exact search method for finding an optimal MAG with respect to given scores. The search combines dynamic programming for learning ancestral graphs and a branch-and-bound search for finding the highest-scoring MAG among the ancestral graphs. We also develop a way to compute and prune Gaussian BIC local scores for the purposes of exact MAG learning. The proposed exact search is empirically shown to often provide higher-quality structures than existing inexact methods.

Paper IV proposes an exact branch-and-bound algorithm for learning optimal causal graphs with respect to given weighted (in)dependence constraints. The search is performed over edge relations and applies linear programming over unsatisfiable cores to compute bounds. Our empirical results show that the presented algorithm is able to learn optimal structures considerably faster than a recent exact method for the problem. A preliminary version of this article received the PGM 2018 Best Student Paper Award [65].

Paper V extends the branch-and-bound algorithm proposed in Paper IV. Firstly, it adds a support for interventional data in the search and non-trivially generalizes the unsatisfiable cores to take advantage of the interventions. Secondly, it proposes the notion of σ-extension to allow the search to use σ-separation [27] as the separation criterion. The paper also makes improvements to an earlier answer set programming (ASP) [50,55] based approach by [27] for learning optimal causal graphs using σ-separation. Empirically we demonstrate that for most problem instances either the extended branch-and-bound search or the improved ASP approach finds an optimal solution faster than the original ASP encoding.

1.2 Author Contributions

All the publications were jointly written by all of the authors. The additional contributions of the present author are listed below.

Paper I: The present author developed the algorithm under the supervision of the co-authors, proposed the on-the-fly score pruning technique for CMSL, developed the way to find relaxedly moral upper bounds, developed and proved the symmetry breaking technique based on preferred vertex ordering, implemented the algorithms and ran all the experiments. The results of the experiments were analyzed together with the co-authors.

Paper II: The present author developed the algorithm under the supervision of the co-authors, implemented the algorithms and ran all the experiments. The results of the experiments were analyzed together with the co-authors.

Paper III: The present author developed the algorithms under the supervision of the co-authors, proposed and proved the theoretically sound way to prune local scores for MAGSL, implemented the algorithms and ran all the experiments. The results of the experiments were analyzed together with the co-authors.

Paper IV: The present author developed the algorithm under the supervision of the co-authors, proposed the ways to speed up the objective function computation by identifying irrelevant edges and unavoidable (non)-colliders, proved all the theoretical results, implemented the algorithm and ran all the experiments. The results of the experiments were analyzed together with the co-authors.

Paper V: The present author proposed the concept of σ-extension and proved its correctness, implemented the support for σ-separation and interventional data and ran all the experiments comparing the running time performance of the different approaches. The results of the experiments were analyzed together with the co-authors.

1.3 Organization of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 overviews Papers I and II, presenting an exact branch-and-bound search and an (in-exact) stochastic local search for learning chordal Markov network structures, and discusses some of the main empirical findings of comparing these approaches to competing methods. Chapter 3 overviews Paper III, presenting an exact search algorithm for learning maximal ancestral graphs from given local scores as well as a way to generate and prune scores for the problem, and evaluates the empirical performance

of the proposed methods. Chapter 4 overviews Papers IV and V, presenting a branch-and-bound approach to learning causal graphs with cycles and latent confounders, and evaluates the practical performance of the algorithm against competing approaches for the problem. Finally, Chapter 5 contains conclusion of the contents of this thesis and further discussion about interesting directions for future work. Proofs and further empirical results are provided in Papers I-V.

Chapter 2

Learning Chordal Markov Networks

Chordal Markov networks (also known as chordal/triangulated Markov random fields, junction/clique trees or decomposable models) are a central class undirected graphical models [18, 44, 47, 95]. Chordality is an appealing property for Markov networks since it makes them equivalent to a type of a Bayesian network in which the parents of each node are adjacent to each other [18, 41]. This allows the models, for example, to use well-justified scoring functions developed for BNs.

This chapter overviews Papers I and II, presenting two algorithmic approaches to learning chordal Markov network structures; an exact branch-and-bound algorithm (Paper I) and an in-exact stochastic local search method (Paper II). Section 2.1 provides an overview to chordal Markov networks and the related optimization problem. In Sections 2.2 and 2.3 we introduce our branch and bound and stochastic local search, respectively. Finally, Section 2.4 discusses empirical results comparing the practical performance of our algorithms against competing exact methods.

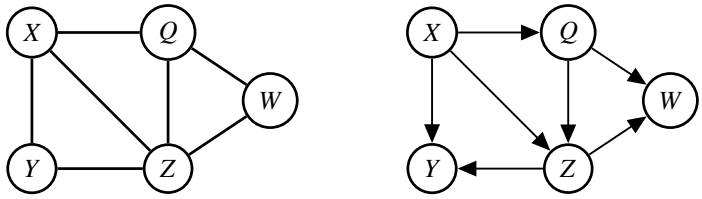


Figure 2.1: *Left:* a chordal Markov network structure. *Right:* an equivalent decomposable DAG.

2.1 Chordal Markov Networks

An undirected graph is *chordal* if for each cycle with four or more nodes in the graph there is an edge connecting two non-consecutive nodes of the cycle. This leads us to the following definition of chordal Markov networks [14].

Definition 1 (Chordal Markov network)**.** *Given a set of random variables V, a chordal Markov network is a pair (G, θ) where*

- *$G = (V, E)$ is the network structure; a chordal undirected graph, and*
- *θ is the network parametrization; a set of factors representing the joint probability distribution of V.*

Example 1. A chordal graph is illustrated in Figure 2.1 (left). Due to the four-node cycle consisting of X, Q, Z and Y, the structure would no longer be chordal if the edge between X and Z was removed.

An alternative view to chordal Markov networks are so-called *decomposable models* which use a *decomposable DAG* as the network structure instead of a chordal undirected graph [18]. A DAG is decomposable if and only if all the parents of each node are adjacent to each other [41]. The case in which some pair of parents is not adjacent is called an *immorality*.

Example 2. A decomposable DAG is illustrated in Figure 2.1 (right). Since both Q and Z are parents of W, the DAG would no longer be decomposable if the edge between Q and Z was removed: in that case Q and Z would form an immorality.

We can observe that the graph on the left in Figure 2.1 is the *skeleton* (undirected version) of the graph on the right. In general, every decomposable DAG is equivalent to some chordal undirected graph and for every undirected chordal graph there exists an equivalent decomposable DAG [18, 41].

Since decomposable models are a special case of Bayesian networks, they can be scored similarly. Suppose s is a decomposable Bayesian scoring function assigning a numerical value $s(v, \mathrm{pa}_G(v))$ to each node v and its *parent set* $\mathrm{pa}_G(v)$ in decomposable DAG $G = (V, E)$. The score of G is then the following [41]:

$$s(G) = \sum_{v \in V} s(v, \mathrm{pa}_G(v)).$$

Note that two Bayesian network structures (DAGs) are *Markov equivalent*, i.e., they encode a same set of conditional independencies, if they share a same skeleton and a same set of immoralities [41]. Since decomposable models have no immoralities, they are Markov equivalent when they share a same skeleton. Here we consider a *score equivalent* scoring criterion which means that all Markov equivalent structures receive a same score [7].

Following Studený and Cussens [85], we define chordal Markov network structure learning as the task of finding a decomposable DAG G^* that maximizes the given Bayesian score function. The skeleton of G^* is then an optimal chordal Markov network structure. Formally,

$$G^* \in \arg\max_{G \in \mathcal{G}} s(G),$$

where $\mathcal{G}$ is the collection of all the relevant decomposable DAGs. In the unrestricted case, $\mathcal{G}$ would contain every possible decomposable DAG over a chosen set of variables. However, sometimes we may reduce the size of $\mathcal{G}$ by limiting the maximum parent set size of the nodes in the graphs, similarly to what is commonly done in Bayesian network structure learning [3, 75, 93].

2.2 Branch and Bound for CMSL

This section overviews Paper I, presenting an exact branch-and-bound approach for CMSL. We begin by providing an overview of the algorithm. After that we explain how to break symmetries in the search (Subsection 2.2.2) and how to compute strong upper and lower bounds (Subsection 2.2.3).

2.2.1 Overview

Branch and bound is an algorithmic paradigm which enables exact solving of optimization problems [57]. Algorithms using this technique implement either a depth-first or breadth-first search and operate on some class of partial solutions which are extended towards feasible solution candidates. A key challenge

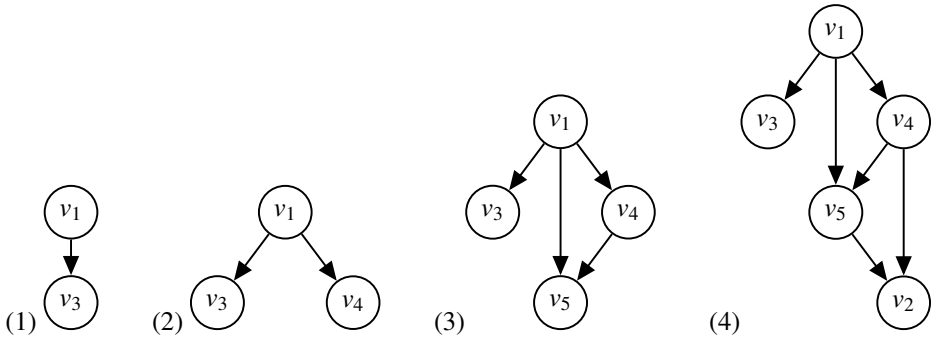


Figure 2.2: An example of how a solution is gradually constructed during the search by adding new sink nodes.

is to find a computationally efficient way to compute tight bounds in the search: the bounds can help us determine which partial solutions cannot lead to an optimal solution, often substantially reducing the number of solution candidates to be considered. In the context of exact approaches to structure learning, the branch-and-bound paradigm has been used successfully for learning Bayesian network structures [86, 87, 93]. Motivated by these advances, here we propose a branch-and-bound approach for learning chordal Markov network structures.

Our search operates on *ordered decomposable DAGs* which are otherwise like plain decomposable DAGs but augmented with a total order over the nodes of the graph. This additional information encodes the order in which the vertices were added to the partial solution during the search and can be used for symmetry breaking (Subsection 2.2.2). Otherwise the order plays no role in the search and we can assume for simplicity that the partial solutions are regular decomposable DAGs.

We can extend partial solutions by choosing a variable which is not yet represented in the graph, adding it as a (sink) node and giving it a parent set. In order to avoid immoralities, only parents which are all adjacent to each other in the graph can be chosen.

Example 3. Figure 2.2 shows how partial solutions can be extended in the search. We start off with a partial solution containing only the node v_1. Then we perform the following steps: (1) Adding v_3 with $\mathrm{pa}(v_3) = \{v_1\}$, (2) adding v_4 with $\mathrm{pa}(v_4) = \{v_1\}$, (3) adding v_5 with $\mathrm{pa}(v_5) = \{v_1, v_4\}$ and (4) adding v_2 with $\mathrm{pa}(v_2) = \{v_4, v_5\}$. The total order of the resulting partial solution is the order in which the nodes were added; that is, v_1, v_3, v_4, v_5, v_2.

The search works as follows. We start off with a partial solution containing only the lexicographically smallest node. Then, in depth-first manner, we try all the ways of adding a new sink node to the current partial solution without causing immoralities. We backtrack in the search by removing the previously added sink node from the current partial solution if (a) all nodes have already been added, (b) the current partial solution violates a symmetry constraint (Subsection 2.2.2), or (c) the upper bound for the current partial solution is too low (Subsection 2.2.3). Whenever all nodes have been added to a partial solution, it serves as a feasible solution candidate to the optimization problem. We keep track of the best (highest-scoring) solution candidate G^* found so far in the search and once the search has finished, G^* will be an optimal solution.

On-the-Fly Score Pruning. Bayesian network structure learning approaches often utilize score pruning techniques [19, 20] in order to reduce the amount of parent set options for each node. Unfortunately these well-known pruning techniques cannot be applied to CMSL as they fail to take immoralities into consideration, meaning that they may prevent us from finding the optimal decomposable DAG.

Instead, we introduce the following *on-the-fly score pruning* for CMSL which prunes scores based on the current partial solution. Suppose the current partial solution can be extended by adding node x with parent set P. Now, if there exists a parent set $\hat{P} \subset P$ such that $s(x, \hat{P}) \leq s(x, P)$, then the parent set option $\hat{P}$ can be omitted for x. This is because adding x with the parents P will enable more parent set choices for future sink nodes than adding x with parents $\hat{P}$.

Example 4. Consider Figure 2.2. When we added v_5 with parents $\{v_1, v_4\}$ in step (3), this allowed us to add v_2 with parents $\{v_4, v_5\}$ in step (4). If we had instead added v_5 with parent set $\{v_1\}$, then we could not have added v_2 with parents $\{v_4, v_5\}$ due to an immorality. Thus, if we also happen to have $s(v_5, \{v_1\}) \leq s(v_5, \{v_1, v_4\})$, then we can prune the parent set choice $\{v_1\}$ for v_5 in step (3). Otherwise, if we have $s(v_5, \{v_1\}) > s(v_5, \{v_1, v_4\})$, then we would have to eventually backtrack back to step (3) in the search and also try adding v_5 with parent set $\{v_1\}$.

2.2.2 Symmetry Breaking

The search space contains solutions which look different from each other but which turn out to be Markov equivalent. For example, if we have a complete graph, we can direct its edges in arbitrary way and always get a decomposable DAG. Yet each of these decomposable DAGs would correspond to a same chordal Markov network structure. In order to make our search efficient, we want to avoid revisiting equivalent solutions. For this purpose we need *symmetry breaking*.

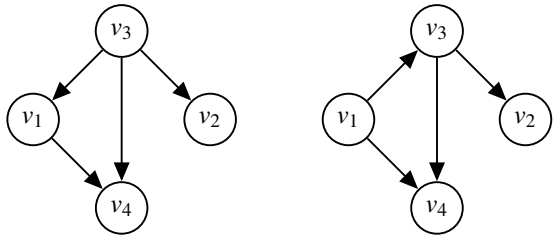


Figure 2.3: Two Markov equivalent decomposable DAGs. The graph on the left violates the preferred vertex order (Definition 2) while the graph on the right does not.

The Bayesian network structure learning approach of van Beek and Hoffmann [93] uses the concept of covered edges [6,7] to prune equivalent graphs from the search space. Here we propose the concept of *preferred vertex order*, a non-trivial generalization of covered edges based on the decomposability of decomposable DAGs.

Definition 2. *Let $G = (V, E)$ be a decomposable DAG. A pair $v_i, v_j \in V$ violates the preferred vertex order in G if the following conditions hold:*

1. *$i > j$,*
2. *$pa_G(v_i) \subseteq pa_G(v_j)$, and*
3. *there is a directed path from v_i to v_j in G.*

We can backtrack in the search whenever some pair of nodes in the current partial solution violates the preferred vertex order. This is due to a theorem proven in Paper I which shows that there always exists a version of the optimal solution in which no pair of vertices violates the preferred vertex order.

Example 5. Consider Figure 2.3. In the graph on the left we have $\mathrm{pa}_G(v_3) \subseteq \mathrm{pa}_G(v_1)$ and $\mathrm{pa}_G(v_3) \subseteq \mathrm{pa}_G(v_4)$, and there are paths from v_3 to both v_1 and v_2. Thus the pairs (v_1, v_3) and (v_2, v_3) violate the preferred vertex order.

In the graph on the right we see that the vertices v_3 and v_2 are the only case where there is a directed path to a lexicographically smaller vertex (conditions (1) and (2) of Definition 2). However, as we have $\mathrm{pa}_G(v_3) \not\subseteq \mathrm{pa}_G(v_2)$, the pair (v_2, v_3) does not violate the preferred vertex order.

While Definition 2 helps us avoid visiting equivalent (but different) solutions, it does not prevent us from constructing an exact same partial solution multiple times. For example in Figure 2.2 we could swap steps (1) and (2) to add $v_1 \to v_4$ before $v_1 \to v_3$ and still end up with the same graph in step (3).

To fix this, we use the same strategy as van Beek and Hoffmann [93] for computing the depth of each vertex in the graph. This information, along with the total order

of the vertices, can be used to unambiguously determine how each partial solution should be constructed in the search. The exact details can be found in Paper I.

2.2.3 Bounds

Computing strong bounds plays an important role in our branch and bound. In addition to upper bounds, we also compute a lower bound in the beginning of the search to act as an initial solution.

We obtain the initial lower bound solution by first finding an optimal Bayesian network structure using a standard dynamic programming algorithm by Silander and Myllymäki [78] and then transforming that structure into a feasible decomposable DAG. We can perform the transformation approximately; the initial lower bound solution does not affect the exactness of the search as long as it represents a feasible solution candidate without cycles and immoralities. Approximation saves time as turning a non-chordal graph into a chordal graph by adding the least number of edges possible is NP-hard [25, 28].

Once we have found the optimal Bayesian network structure using the dynamic programming algorithm [78], we also obtain as a by-product the scores of the optimal *Bayesian network extensions* for all the subsets of the vertices. That is, given any subset of vertices $U \subseteq V$, we have a cache telling us how we could optimally extend a BN structure over U by adding nodes $V \setminus U$. This insight provides us with an efficient way of computing upper bounds in our branch and bound. If the score of the current partial solution plus its optimal Bayesian network extension is not higher than that of the best solution found so far in the search, then we know that the current partial solution cannot lead to an improvement and so we can backtrack in the search.

The upper bounds obtained via the BN extensions can be quite loose since they do not account for immoralities in any way. For this reason we developed an alternative dynamic programming algorithm which finds a *relaxedly moral* extension for the current partial solution. Here relaxedly moral means that immoralities are not allowed to occur between any parents which are both in the partial solution. On the other hand immoralities in which either of the parents is in the extension are allowed.

Example 6. Figure 2.4 shows a partial solution (consisting of nodes v_1, v_3, v_4, v_6, v_7 and v_8) which has been extended with a relaxedly moral extension consisting of nodes v_2 and v_5. Since both v_7 and v_2 are parents of v_5, and there is no edge between v_2 and v_7, the graph contains an immorality. This is allowed since one of the parents (namely v_2) is part of the extension.

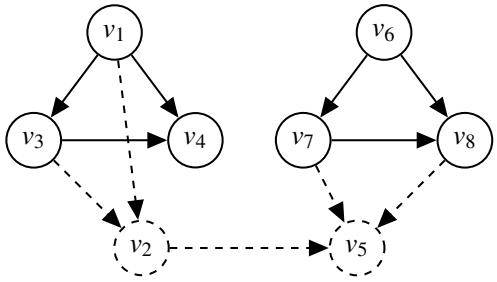


Figure 2.4: An example of a relaxedly moral upper bound solution. The vertices and edges with solid lines represent a partial solution and the vertices and edges with dashed lines are its relaxedly moral extension.

While the relaxedly moral extensions can provide tighter upper bounds than the BN extensions, they come with an additional computational cost. Unlike with the BN extensions, the relaxedly moral extensions depend on the edges in the decomposable DAG and thus have to be computed individually for each partial solution.

2.3 Stochastic Local Search for CMSL

This section overviews Paper II, presenting a stochastic local search for chordal Markov network structure learning. Subsection 2.3.1 provides an introduction to the algorithm, and Subsection 2.3.2 explains the neighborhoods used in the search.

2.3.1 Overview

Local search is an in-exact algorithmic paradigm for solving optimization problems [31] which has turned out to be very effective for learning Bayesian network structures [45,46]. A key challenge in developing practically effective local search methods is to define meaningful *neighborhoods* which group similar solution candidates ("neighbors") together. The algorithm explores the search space by continuously moving from a solution candidate to one of its neighbors and hence the chosen neighborhood type greatly affects how good solutions the search will find. To avoid slowing down the search, the neighborhoods should be designed in such way that the highest-scoring neighbor can be found reasonably quickly. However, instead of always moving greedily to the highest-scoring neighbor, stochasticity can be used in the neighbor selection process to allow wider exploration of the search space.

Our search uses undirected graphs as solution candidates. When scoring a solution, we turn it temporarily into decomposable DAG by directing its edges using a standard polynomial-time algorithm by Rose *et al.* [71]. The algorithm will also reveal whether the graph is chordal; we will dismiss all non-chordal graphs in the search.

The search works as follows. We start off by computing an optimal tree-shaped Bayesian network structure using the polynomial-time Chow-Liu algorithm [9]. The skeleton of this structure is always chordal and acts as our initial solution. Then we make slight gradual changes to the solution by repeatedly applying neighborhood operations (Subsection 2.3.2). This allows us to move through the search space and visit different solution candidates. We also keep track of the highest-scoring graph G^* we have encountered so far in the search.

We also employ random restarts whenever the search seems to be stuck. For this purpose we keep track of the highest-scoring solution $\hat{G}$ found since the last (re)start. If a certain number of steps are taken with no improvements to $\hat{G}$, we restart the search by using a randomly generated chordal graph as the new starting point.

On-the-Fly Score Computation. Exact CMSL approaches compute all the possible local scores prior to starting the search [14, 38, 39, 40]. This can be very time consuming given that there are exponential number of local scores with respect to the number of variables in the dataset. For this reason our stochastic local search employs *on-the-fly score computation*: a local score is only computed if (and when) it is needed in the search for scoring a solution. We show in Paper II that this strategy significantly reduces the amount of local scores that are needed to be computed for the search.

2.3.2 Neighborhoods

A *neighborhood* is a set of new solution candidates (undirected chordal graphs) which arise from applying some neighborhood operation (structural change) to the current solution. We define three types of neighborhoods based on cliques, nodes and edges.

Edge-specific neighborhood $N_E(G, v)$: Contains all the solutions that are obtained from $G = (V, E)$ by either adding or removing an edge to $v \in V$.

Example 7. In Figure 2.5, we form the edge-specific neighbors of G with respect to node Y by adding edge $Y - Z$, adding edge $Y - W$ and removing the edge $Y - X$. Note that the first neighbor (after adding $Y - Z$) is not chordal and would thus be dismissed in the search.

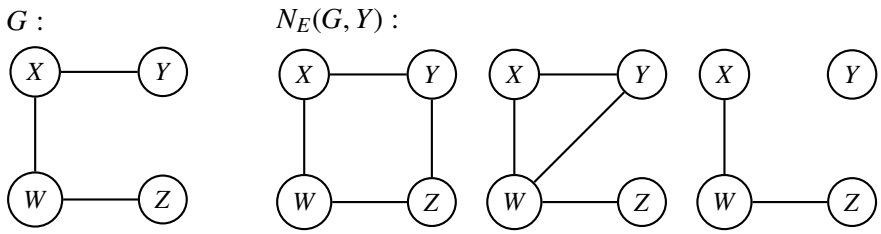


Figure 2.5: Edge-specific neighborhood for graph G wrt node Y.

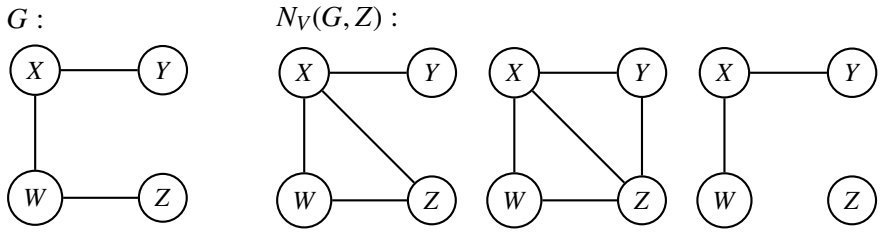


Figure 2.6: Node-specific neighborhood for graph G wrt node Z.

The two remaining neighborhood types target the *maximal cliques* of the graph. A clique is a set of pairwise adjacent vertices. Clique C is maximal if the graph does not contain a clique $\hat{C}$ for which $C \subset \hat{C}$. Extracting the maximal cliques from the graph is straightforward using the algorithm by Rose *et al.* [71].

Node-specific neighborhood $N_V(G, v)$: Contains all the solutions that are obtained from $G = (V, E)$ by adding or removing $v \in V$ to a maximal clique in G.

Example 8. In Figure 2.6, the maximal cliques of G are $\{X, Y\}$, $\{X, W\}$ and $\{W, Z\}$. We construct the node-specific neighbors of G with respect to node Z by

- adding Z to clique $\{X, W\}$ to form clique $\{X, W, Z\}$,
- adding Z to clique $\{X, Y\}$ to form clique $\{X, Y, Z\}$, and
- removing Z from clique $\{Z, W\}$.

Clique-specific neighborhood $N_C(G, C)$: Contains all the solutions that are obtained from G by adding or removing a node to maximal clique C in G.

Example 9. In Figure 2.7, we construct the clique-specific neighbors of G with respect to the maximal clique $\{X, W\}$ by

- adding Z to clique $\{X, W\}$ to form clique $\{X, W, Z\}$,
- adding Y to clique $\{X, W\}$ to form clique $\{X, W, Y\}$, and
- removing X (or W) from clique $\{X, W\}$.

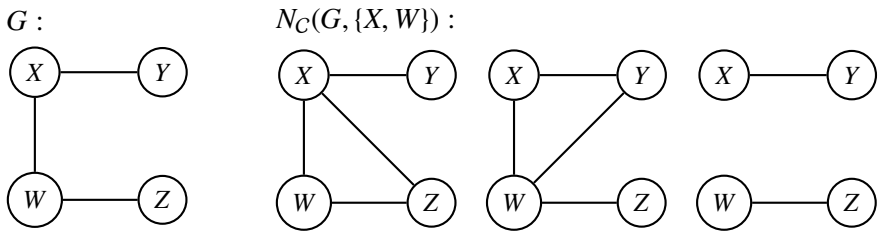


Figure 2.7: Clique-specific neighborhood for graph G wrt clique $\{X, W\}$.

A stochastic neighborhood for solution G can be constructed in the following way. First we choose a random node v and random maximal clique C in G. Then we consider all the feasible (chordal) solutions in $N_V(G, v), N_E(G, v)$ and $N_C(G, C)$, and the highest-scoring of these graphs is chosen as the starting point for the next iteration in the search.

2.4 Empirical Evaluation

This section provides an overview of some of the main empirical findings on the performance of our two CMSL approaches. See Papers I and II for more detailed results.

We implemented both the branch and bound and the local search in C++ and will refer to these implementations as *BBMarkov* and *LSMarkov*, respectively. Both implementations are available in open source online.

In addition to comparing the implementations to each other, we also investigate how they compare against competing exact CMSL approaches GOBNILP [85] and Junctor [39, 40]. The declarative approach presented by Corander *et al.* [14] is not evaluated here as it is considerably slower than GOBNILP and Junctor, being only able to solve instances with up to around eight variables. For the experiments we used the BDeu score with equivalent sample size 1 and a number of discrete real-world datasets [93, 97] (problem instances) commonly used for benchmarking structure learning algorithms.

Exact Learning. Figure 2.8 shows how quickly BBMarkov, GOBNILP and Junctor learned optimal chordal Markov network structures for instances with different number of variables n from 11 up to 17. Each algorithm was given a 1-hour time limit to solve each instance to optimality and no structural restrictions are enforced on the solutions.

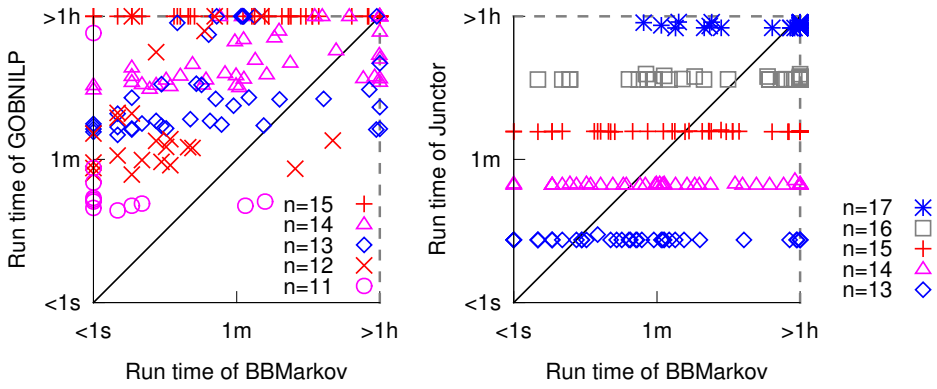


Figure 2.8: Comparison of the running time performance of BBMarkov to GOBNILP (left) and Junctor (right). The color of each marker indicates the number of variables n in the corresponding instance.

Figure 2.8 (left) shows that BBMarkov outperforms GOBNILP in terms of time required to solve instances. In fact, GOBNILP seems to be unable to scale beyond 15 variables within the time limit.

Junctor exhibits more solid performance compared to BBMarkov (Figure 2.8, right). Due to its dynamic programming strategy, the algorithm requires approximately same amount of time to solve any instance of same n, explaining the horizontal alignment of the markers in the plot. For this reason Junctor is able to solve any 17-variable instance within one hour whereas BBMarkov can suffer timeouts even for some smaller instances. On the other hand, Junctor has a $\Omega(4^n)$ time complexity for an n-variable instance, meaning that solving an instance will take approximately four times longer for every new variable introduced. It also requires tremendous amount of memory with its $\Omega(3^n)$ space complexity. These factors set hard limits on what can be solved with the algorithm.

Table 2.1 shows the best-case performance of BBMarkov. Here we used a 24-hour time limit and 32-GB memory limit. We can see that Junctor is unable to solve any 20-variable instances due to its high memory consumption. Furthermore, if we consider how much time it required for 19 variables, we can conclude that it could not have solved 20 variables within 24 hours even with unlimited memory. Hence BBMarkov appears to be the only exact CMSL approach able to scale up to 20 variables when no special structural restrictions are enforced on the solutions.

Stochastic Local Search. Unlike the exact approaches, LSMarkov is unable to prove the optimality of the solutions it finds. For this reason we investigated the

Table 2.1: Examples of best-case performance of BBMarkov. The numbers in parentheses show how much time it required to find the optimal solutions.

		Running times (s)		
Dataset	n	**BBMarkov**		**Junctor**
alarm	18	1462	(315)	12477
	19	10274	(2028)	52130
	20	49610	(50)	*memout*
Heart	18	162	(85)	11179
	19	1186	(698)	50296
	20	15501	(13845)	*memout*
insurance100	18	494	(113)	11334
	19	17125	(15150)	50014
	20	*timeout*	(*timeout*)	*memout*
hailfinder500	18	2543	(1348)	12422
	19	13749	(6418)	53108
	20	33503	(25393)	*memout*
mildew1000	18	1751	(1278)	11294
	19	33160	(27421)	52572
	20	*timeout*	(*timeout*)	*memout*
water100	18	590	(244)	12244
	19	6581	(6187)	52575
	20	61152	(54806)	*memout*

accuracy of LSMarkov empirically on a set of 23 instances which were marginalized to 17 variables each. For 22 out of the 23 instances, LSMarkov was able to find an optimal solution, i.e., a chordal Markov network structure which matched the one found by Junctor. Moreover, LSMarkov required considerably less time to find the optimal solutions compared to the exact approaches.

As stated earlier, the exact approaches only scale up to around 20 variables without enforcing structural restrictions on the solutions. A known way to circumvent this is to limit the maximum clique size in the found graphs [39, 40]. This allows for solving larger instances but can also worsen the accuracy (score) of the solutions. On the other hand, due to the on-the-fly score computation strategy of our stochastic local search, LSMarkov is able to scale much further than 20 variables even without enforcing structural restrictions on the solutions.

We solved the 23 instances again, this time without limiting the number of variables in each dataset. The exact approaches were enforced a 10-hour time limit and a maximum clique size limit of 3, while LSMarkov was allowed only 15 *minutes* of running time on each instance but no limit on the maximum clique size in its solutions.

Table 2.2: Comparison of the CMSL approaches on benchmark datasets. Here LSMarkov was ran without structural restrictions while the other algorithms were tested enforcing a maximum clique size $cs \leq 3$. Highest scores and among those the shortest time are in bold.

		LSMarkov		**GOBNILP** $cs \leq 3$		**BBMarkov** $cs \leq 3$	
Dataset	n	**Score**	**Time (s)**	**Score**	**Time (s)**	**Score**	**Time (s)**
autos	26	**-1513.2**	**100**	-1641.8	73	-1866.7	17
water10000	26	-128910.9	769	**-128909.2**	**17**	-130327.0	18
insurance10000	27	**-134175.5**	**290**	-134701.2	92	-159329.4	34
steel	28	**-18604.4**	**425**	-19908.1	561	*no*	-
horse	28	**-4503.0**	**513**	-4511.9	19	*no*	-
flag	29	**-2720.0**	**339**	-2747.8	47	*no*	-
Epigenetics	30	**-177782.8**	**854**	-188915.2	230	*no*	-
wdbc	31	**-6654.7**	**602**	-6771.5	1762	*no*	-
mildew10000	35	-454360.8	545	**-454312.8**	**133**	*no*	-
soybean	36	**-2932.2**	**715**	-3332.8	1354	*no*	-
alarm10000	37	**-105155.3**	**408**	-105424.7	378	*no*	-
bands	39	**-5099.6**	**350**	-5145.1	293	*no*	-
connect-4	43	**-1008451.1**	**171**	-1043546.0	573	*no*	-
spectf	45	-8055.0	355	**-8047.4**	**3067**	*no*	-
sponge	45	**-1788.1**	**89**	-1813.1	4264	*no*	-
barley5000	48	**-266863.3**	**537**	-269916.7	1032	*no*	-
hailfinder10000	56	**-497216.1**	**758**	-497784.1	9134	*no*	-
lung-cancer	57	**-1248.9**	**262**	-1366.2	8024	*no*	-
promoters	58	**-8326.5**	**3**	**-8326.5**	1693	*no*	-
carpo10000	60	**-173880.7**	**765**	-176671.8	9344	*no*	-
kddcup	60	**-72971.4**	**207**	*no*	-	*no*	-
dota2	115	**-442078.8**	**20**	*no*	-	*no*	-
msweb	235	**-51596.0**	**881**	*no*	-	*no*	-

The results are shown in Table 2.2. Junctor is omitted from the table as it was unable to scale up to $n \geq 26$ variables due to its space complexity even when limiting the maximum clique size. Similarly BBMarkov is unable to provide any solutions for $n \geq 28$ variables due to its heavy memory consumption. Meanwhile GOBNILP seems to be the best out of the exact approaches in this setting, being able to provide solutions (not necessarily optimal) for instances with up to $n = 60$ variables. However, for 20 out of the 23 instances, LSMarkov was able to find a better (higher-scoring) solution within 15 minutes than the exact approaches did within 10 hours.

Finally, Figure 2.9 shows the impact of the neighborhood types of Section 2.3.2 in the local search. We ran the algorithm with every combination of the three neighborhoods in order to investigate how good solutions (in terms of score) can be obtained for the instances in each case. Here C, V and E denote clique-specific, vertex-specific and edge-specific neighborhoods, respectively. For example "CV" uses both clique- and vertex-specific neighborhoods but not the edge-specific

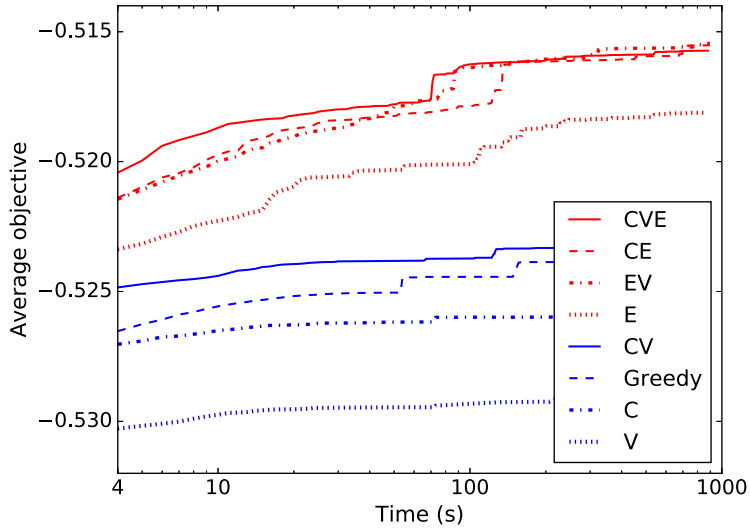


Figure 2.9: The impact of neighborhoods on LSMarkov with respect to the average objective value; that is, the sum of the best scores during LSMarkov search at different time points divided by the number of instances (23) and by the number of variables and samples in each instance. The plot starts at 4 seconds as the earliest point when an initial solution was obtained for every instance.

neighborhood. We can see from the figure that none of the neighborhood types on their own provide the best performance for the local search: thus all the three neighborhood types are important.

Figure 2.9 also shows the performance of a greedy version of the search. The greedy version works otherwise like "CVE" but it does not select a random vertex v and maximum clique C for constructing a stochastic neighborhood. Instead, it considers *all* vertices and maximum cliques in the current solution, making the search greedily choose the highest-scoring solution out of all the possible neighbors in each step. The figure shows that the stochastic strategy with all neighborhood types ("CVE") performs much better than the greedy version of the search.

Chapter 3

Learning Maximal Ancestral Graphs

A drawback of Markov networks and Bayesian networks is that they do not account for *latent confounding*, a situation where an unobserved variable is a common cause of two observed variables [80]. Thus inference based on those model classes must assume that the common causes of the observed variables are observed themselves, known as the *causal sufficiency*, which is often unrealistic in real-world settings [80].

Maximal ancestral graphs (MAGs) are a theoretically appealing generalization of Bayesian networks to account for latent confounding [67]. MAGs have been used extensively in causal inference because they can represent marginalizations of Bayesian networks and retain many of their important properties [37, 62, 70, 80, 99, 100]. However, unlike in the case of BNSL and CMSL, exact approaches to learning MAGs have not been developed, despite the potential gains in accuracy.

This chapter overviews Paper III, presenting the first exact algorithm for learning maximal ancestral graphs from given local scores. Section 3.1 provides an introduction to maximal ancestral graphs and the related learning problem. In Section 3.2 we propose how to perform exact search from given local scores as well as how to generate and prune scores for MAG structure learning. Finally, in Section 3.3 we discuss empirical results of our score computation and pruning methods, and compare our exact search against relevant in-exact approaches for the problem.

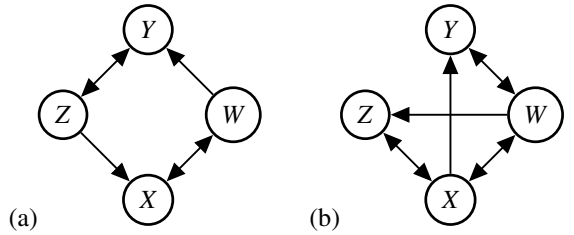


Figure 3.1: (a) A maximal ancestral graph, and (b) a graph which is not a MAG due to an inducing path between non-adjacent Z and Y.

3.1 Maximal Ancestral Graphs

We begin by overviewing key concepts needed in the definition of maximal ancestral graphs. An *almost directed cycle* is a directed path between nodes which are connected by a bidirected edge; i.e., $X \to \ldots \to Y$ with $X \leftrightarrow Y$. A *collider* on a path is a node that has the adjacent edges on the path pointing towards the node, i.e., either $\to Z \leftarrow$, $\leftrightarrow Z \leftrightarrow$, $\to Z \leftrightarrow$ or $\leftrightarrow Z \leftarrow$. An *inducing path* is path $X \cdots Y$ where each vertex (except for X and Y) is a collider and there is a directed path from each collider to either X or Y.

Following the definition by Zhang [99], the maximal ancestral graphs considered in this thesis can contain both directed and bidirected edges but not undirected edges. The bidirected edges $X \leftrightarrow Y$ represent latent confounders: i.e., structures such as $X \leftarrow L \to Y$ where L is an unobserved common cause of both X and Y.

Definition 3 (Maximal ancestral graph)**.** *A mixed graph $G = (V, E)$ containing a combination of directed $X \to Y$ and bidirected $X \leftrightarrow Y$ edges is a maximal ancestral graph (MAG) if*

- *G is* ancestral*: contains neither directed nor almost directed cycles; and*
- *G is* maximal*: there is an inducing path between nodes X and Y only if X and Y are adjacent.*

Example 10. The graph illustrated in Figure 3.1 (a) is a MAG: it contains neither directed nor almost directed cycles and every inducing path is between adjacent nodes. Meanwhile the graph shown in Figure 3.1 (b) is not a MAG since there is an inducing path $Z \leftrightarrow X \leftrightarrow W \leftrightarrow Y$ in which the colliders X and W are ancestors of the non-adjacent endpoints Z and Y of the path.

The maximality and ancestrality conditions of MAGs allow for asymptotically consistent scoring via the Gaussian BIC scoring function [67, 92]. The score for MAG $G = (V, E)$ is

$$s(G) \quad = \quad \ln L_G(\hat{\theta}) - (2|V| + |E|)/2 \cdot \ln N, \tag{3.1}$$

where L_G is the multivariate Gaussian likelihood function, N is sample size and $\hat{\theta}$ are maximum likelihood parameters for the linear Gaussian model over G [92]. The number of parameters accounts here for the mean and variance for each vertex, and one coefficient per edge.

Since an inducing path between two nodes allows for the corresponding variables to be dependent given any conditioning set, maximality ensures that a non-adjacency always implies some conditional independence relation. Thus Markov equivalent MAGs share same adjacencies [81]. Moreover, since Markov equivalent MAGs can also represent the same multivariate Gaussians distributions [67, Corollary 8.19], the BIC score is score equivalent.

Another key feature of the Gaussian BIC score is that it factorizes in terms of *c-components* [69, 88]. A c-component is a subgraph in which all the nodes are strongly connected via bidirected edges. In other words, between every pair of nodes in the component there is a path consisting entirely of bidirected edges.

Let C be a c-component over nodes $X_1, \ldots, X_n$. We denote the parents of C in MAG G as a list $\mathrm{pa}_G(C) = (\mathrm{pa}_G(X_1), \ldots, \mathrm{pa}_G(X_n))$. Furthermore, we define a *local c-component* to be a pair $(C, \mathrm{pa}(C))$ consisting of a c-component and a list of parents for its nodes.

Example 11. The MAG in Figure 3.1 (a) consists of two local c-components: $Z \leftrightarrow Y$ with W as the parent of Y, and $X \leftrightarrow W$ with Z as the parent of X. That is, $(Z \leftrightarrow Y,\ (\emptyset, \{W\}))$ and $(X \leftrightarrow W,\ (\{Z\}, \emptyset))$.

Similarly to Tsirlis *et al.* [92], we can individually score each local c-component $(C, \mathrm{pa}_G(C))$. To achieve this, suppose $\hat{G}$ is the MAG *induced* by $(C, \mathrm{pa}_G(C))$: that is, $\hat{G}$ consists of C and its parents. Furthermore, let G' be an empty MAG consisting of the parents of C not in C. The score of the component is then $s(C, \mathrm{pa}_G(C)) = s(\hat{G}) - s(G')$. This allows for scoring a MAG G over c-components $C_1, \ldots, C_m$ as follows [92]:

$$s(G) = \sum_{i=1}^{m} s(C_i, \mathrm{pa}_G(C_i)). \tag{3.2}$$

We define *maximal ancestral graph structure learning* (MAGSL) as the problem of finding a MAG G^* that maximizes the given Gaussian BIC local scores. Formally,

$$G^* \in \arg\max_{G \in \mathcal{G}} s(G),$$

where $\mathcal{G}$ is the collection of all the relevant MAGs. In the unrestricted case, $\mathcal{G}$ would contain every possible MAG over a chosen set of variables. However, some structural restrictions on $\mathcal{G}$ are required in order to make score computation and exact learning possible for non-trivial instances. We use c to denote the maximum number of nodes per c-component, and p to denote the maximum number of parent relations per c-component. That is, each c-component C has to satisfy $|\bigcup_{x \in C} x| \leq c$ and $\sum_{x \in C} |\text{pa}(x)| \leq p$. This means that configuration $c = 1, p$ would be equivalent to Bayesian network structure learning with maximum number of parents per node being p.

3.2 Exact Search for MAGSL

This section introduces an exact algorithm for learning optimal maximal ancestral graphs with respect to given Gaussian BIC scores as proposed in Paper III. Subsection 3.2 provides an overview of the algorithm and Subsections 3.2.2 and 3.2.3 explain how local scores can be computed and pruned for MAGSL.

3.2.1 Exact Search

Our exact search is divided into two components: a dynamic programming algorithm for learning optimal *ancestral* graphs (AGs) which are otherwise like MAGs but without the maximality requirement; and a branch-and-bound algorithm for finding the highest-scoring MAG using the found AGs.

Much like in the case of our branch and bound for CMSL (Section 2.2), we consider partial solutions (graphs) which are extended during the search by adding new nodes. However, here the partial solutions are AGs, and instead of adding nodes one at a time, we add entire local c-components which can consist of multiple nodes. The reason for this is that even though BNSL and CMSL algorithms can make use of the fact that removing a sink node from an optimal structure results in an optimal structure over the remaining nodes, the same does not hold true in general for (M)AGs as is demonstrated with a counterexample in Paper III.

It might be tempting to generalize the concept of "removing sink nodes" to removing sink c-components (i.e., components which do not have outward edges to

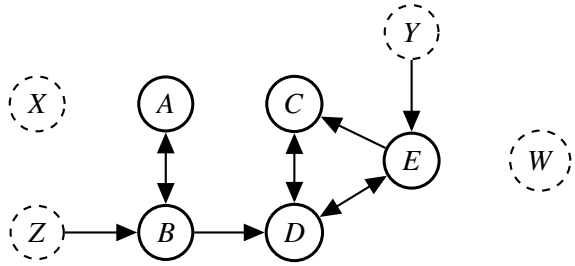


Figure 3.2: An example scenario in the search. A partial solution is shown with solid lines and the remaining nodes of the optimization problem shown with dashed lines.

other components). Unfortunately, we cannot rely on such property when developing an exact MAGSL approach since not all (M)AGs (e.g. Figure 3.1) contain a sink c-component. Thus the local c-components added to partial solutions must be allowed to have both inward and outward edges to other components.

We say that the *remaining nodes* U for a given partial solution are all the relevant nodes of the optimization problem which are not part of the partial solution. A partial solution is a feasible solution candidate to the problem when $U = \emptyset$. For each partial solution we keep track of the *reach* R_U and *almost reach* A_U of the remaining nodes U. That is, X reaches Y, denoted with $(X, Y) \in R_U$, if there is a directed path from X to Y. Similarly X almost reaches Y, denoted with $(X, Y) \in A_U$, if there exists a path $X \cdots Y$ which (1) contains exactly one bidirected edge, and (2) the path could become a directed path from X to Y if the bidirected edge was changed into a directed one. Clearly, a graph has a directed cycle if some node reaches itself, and an almost directed cycle if some node almost reaches itself. Therefore R_U and A_U define how an AG can be extended during the search while maintaining ancestrality.

Example 12. Figure 3.2 shows a partial solution over nodes A, B, C, D, E. Two local c-components have been added to it; $(A \leftrightarrow B, (\emptyset, \{Z\}))$ and $(C \leftrightarrow D \leftrightarrow E, (\{E\}, \{B\}, \{Y\}))$. No local c-components have yet been chosen for the remaining nodes $U = \{X, Y, Z, W\}$. We have $R_U = \{(Z, B), (Z, D), (Y, E), (Y, C)\}$ and $A_U = \{(Z, A), (Z, E), (Z, C), (Y, D)\}$. Therefore we cannot extend the partial solution in any way which would, for example, involve edge $Z \leftarrow D$ as that would form a directed cycle. We also cannot add edge $Y \leftarrow D$ as this would form an almost directed cycle. The partial solution could be extended in the search e.g. by forming a c-component $X \leftrightarrow Y$ with parents $\mathrm{pa}(X) = \{A, D\}$ and $\mathrm{pa}(Y) = \{Z, W\}$.

Computing AG Extensions. As discussed in Subsection 2.2.3, solving an optimization problem via dynamic programming has the added benefit of also finding

solutions for all the subproblems. Therefore by learning an optimal AG over the relevant variables of the problem we also learn the optimal *AG extensions* for each subset of the variables. Here an optimal AG extension refers to an optimal way to extend a given (M)AG such that the resulting (M)AG contains neither directed nor almost directed cycles.

Our dynamic programming starts off with an empty partial solution (no nodes) and then in depth-first manner tries to extend it with all combinations of local c-components while maintaining ancestrality. This allows us to see which combinations (i.e., AG extensions) yield the highest scores.

To make the procedure efficient, we keep cached the highest score (corresponding to the best AG extension) found for each triple (U, R_U, A_U). If a partial solution has same (U, R_U, A_U) values as some previously encountered solution, then we simply fetch the already-computed value from cache. For example the partial solution in Figure 3.2 would still have the same (U, R_U, A_U) values even if parents $C \leftarrow Y$ and $D \leftarrow A$ were introduced. We also keep the reachability relations succinct by enforcing for all nodes X, Y that $(X, Y) \notin A_U$ if $(X, Y) \in R_U$: the latter is already enough for retaining ancestrality. This makes partial solutions more likely to share same (U, R_U, A_U) values.

Finding Optimal MAGs. The R_U, A_U relations used in the proposed dynamic programming are not sufficient for keeping track of inducing paths or ensuring maximality. For this reason we present a branch-and-bound algorithm for finding the highest-scoring MAG using the AGs found via dynamic programming.

The search works as follows. Like previously, we start off with an empty partial solution and try adding different local c-components to it one by one, this time maintaining both ancestrality *and* maximality along the way. For each partial solution encountered in the search, we construct the optimal AG extension for the current (U, R_U, A_U) values. This can be done in polynomial time once the dynamic programming cache has been built. Then we extend the partial solution with the AG extension to create AG G' over all nodes. With it we can perform the following reasoning to make the search efficient:

- If G' is maximal, it represents the highest-scoring MAG achievable for the current partial solution and thus we can backtrack in the search. We keep track of the highest-scoring MAG G^* found this way.
- Regardless of whether G' is maximal, we can backtrack in the search if its score is not better than that of G^*.

Once the search has finished, G^* is an optimal MAG. Note that while this branch-and-bound procedure is vital for ensuring the correctness of the overall search, in practice the optimal MAG can often be straightforward to obtain via the dynamic

programming cache. Hence the time spent in the branch-and-bound phase is often negligible compared to the time spent in the dynamic programming.

Finally, we want to prevent a situation where the search would create a same solution multiple times by adding same c-components in different order. To do this, we enforce that each new c-component to be added to a partial solution must contain the lexicographically smallest node in U. Note that this does not enforce any ordering among the nodes in the found (M)AGs.

3.2.2 Scoring

Score-based structure learning approaches [35, 39, 78, 85, 93] often require computing local scores from data before the structure learning can begin. Similarly we need a framework for generating all the needed c-component scores for exact MAGSL.

Let V be the set of relevant variables of the problem instance, c be the maximum c-component size limit and p be the parent set limit. We can produce the corresponding local c-component scores in the following way:

1. Iterate every possible mixed graph, containing at most c nodes from V and at most p parents, such that all the nodes in the graph are strongly connected via bidirected edges.
2. Discard any graph from the previous step which is not maximal or ancestral. The remaining graphs correspond to local c-components containing only internal parents, i.e., parents within the component.
3. Iterate every possible way of extending the local c-components from the previous step by adding external parents. That is, if a c-component consists of nodes $U \subseteq V$ and has p' parents, iterate all the ways of adding at most $p - p'$ parents from $V \setminus U$ to the component.

Similarly to M3HC [92], we use *residual iterative conditional fitting* (RICF) [22] to compute the maximum likelihood estimates needed for the Gaussian BIC scores of each local c-component. The strong convergence properties of RICF state that it converges to an accumulation point from any given starting point, and all accumulation points yield the same value for the likelihood function [22, Theorem 13]. Note that there can exist multiple local optima especially for small sample sizes [1, 23]. In practice, RICF usually converges to only a few different likelihood values when ran from random starting points for local c-components of limited size c. We show empirically in Paper III that after applying our pruning scheme (Subsection 3.2.3), only 0.28% of the remaining local scores had two ICF

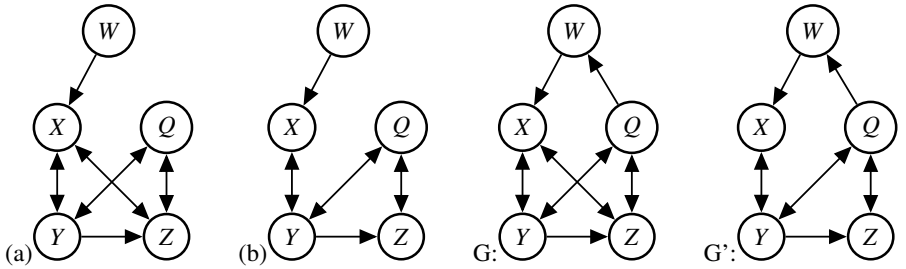


Figure 3.3: Replacing local c-component (a) with (b) in MAG G results in graph G' which is not a valid MAG.

accumulation points for $c = 2$, and 0.45% had two or three ICF accumulation points for $c = 3$.

3.2.3 Score Pruning

The scalability of exact score-based approaches for Bayesian network structure learning is made possible by score pruning techniques [19, 20]. Here we propose rules for pruning local c-component scores for MAG structure learning.

The goal is to identify local c-components which are guaranteed not to be a part of an optimal MAG. The exact search then saves time by not having to consider (partial) solutions which feature such components. Suppose (C, P) and (C', P') are c-components such that C and C' are over a same set of nodes. The component (C, P) can be pruned if in every MAG G containing (C, P) we can always replace (C, P) with (C', P') and the resulting graph G' is a MAG with $s(G) < s(G')$. Note that $s(C) < s(C')$ implies $s(G) < s(G')$ due to how the score factorizes (Equation 3.2).

We begin by explaining why a trivial generalization of the standard pruning used in BNSL is not applicable to MAGSL. Given variable x and parent set P, local score $s(x, P)$ can be discarded in BNSL if we have $s(x, P) \leq s(x, P')$ for some parent set $P' \subset P$. Now, suppose C and C' are c-components over a same set of nodes. A straightforward generalization of the BNSL pruning would be to discard local score $s(C, \mathcal{P})$ if $s(C, \mathcal{P}) \leq s(C', \mathcal{P}')$ and the mixed graph induced by $(C', \mathcal{P}')$ is a subset of the mixed graph induced by $(C, \mathcal{P})$. However, such pruning rule is not sound, because removing bidirected edges can violate the maximality condition of MAGs, as we see in the following example.

Example 13. Consider Figure 3.3. Pruning local c-component (a) due to a higher score for (b) might prevent us from finding optimal MAG G, as replacing (a) with (b) in G would result in graph G' which is not a valid MAG. Note that G' violates

the maximality condition by containing inducing path $X \leftrightarrow Y \leftrightarrow Q \leftrightarrow Z$ between non-adjacent nodes X and Z.

To form theoretically solid pruning rules for MAGSL, we propose the notion of *maximality-preserving pair* which denotes node pairs which cannot have a non-trivial (i.e., longer than single edge) inducing path between them. Thus, if a mixed graph is non-maximal, it cannot be due to a missing edge between a maximality-preserving pair. Paper III shows two ways for a pair $\{x, y\}$ to be maximality-preserving: either (1) all paths between x and y containing only bidirected edges are less than four nodes long, or (2) we have $\mathrm{pa}(x) \subset \mathrm{pa}(y)$. This leads us to the following pruning rule for MAGSL which is shown to be theoretically sound in Paper III.

Pruning Rule 1. *Let C and C' be c-components over a same set of nodes and let their parent set lists be P and P', respectively. The score $s(C', P')$ can be pruned if the following conditions hold:*

1. $s(C, P) \geq s(C', P')$;
2. *the mixed graph induced by (C, P) is a subgraph of the mixed graph induced by (C', P'); and*
3. *for all $X, Y \in C'$, if $X \leftrightarrow Y$ exists in C' but not in C, then $\{X, Y\}$ must be a maximality-preserving pair in the mixed graph induced by (C, P).*

We can achieve even more powerful pruning due to the following simple observation: when bidirected edges are deleted from a MAG (between maximality-preserving pairs), some c-components may get cut in half in the process. This leads to the following pruning rule (shown to be theoretically sound in Paper III) which states that a local c-component should be pruned if it can be partitioned into smaller local c-components which achieve a higher (or equal) score together.

Pruning Rule 2. *Let $C_1, \ldots, C_n$ and C' be c-components with parent set lists $P_1, \ldots, P_n$ and P', respectively. Suppose the nodes in $C_1, \ldots, C_n$ partition the nodes in C' to distinct subsets. The score $s(C', P')$ can be pruned if the following conditions hold:*

1. $\sum_i s(C_i, P_i) \geq s(C', P')$;
2. *the union of the mixed graphs induced by $(C_1, P_1) \ldots (C_n, P_n)$ is a subgraph of the mixed graph induced by (C', P'); and*
3. *for all $X, Y \in C'$, if $X, Y \in C_i$ and $X \leftrightarrow Y$ exists in C' but not in C_i, then $\{X, Y\}$ must be a maximality-preserving pair in the mixed graph induced by (C_i, P_i).*

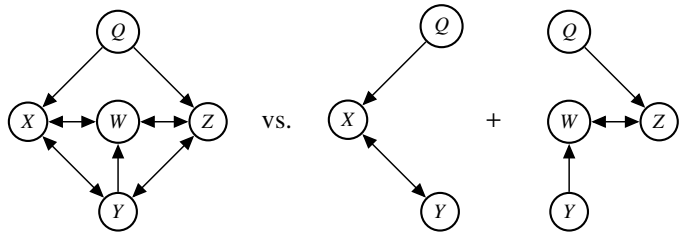


Figure 3.4: The local score for the component on the left can be pruned if the sum of scores for the two components on the right is higher.

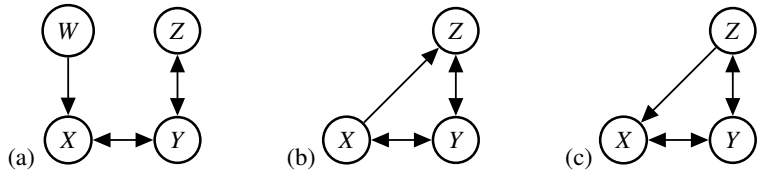


Figure 3.5: Local c-component (a) is neither a subgraph nor supergraph of local c-components (b) and (c). Yet (a) could be pruned if either (b) or (c) have higher score.

Example 14. Figure 3.4 shows an example of applying Pruning Rule 2. The c-component over X, W, Y, Z with external parent Q for X and Z (left) can be pruned if the c-components over X, Y and W, Z (center, right) have a higher sum of scores. Note that removing edges $X \leftrightarrow W$ and $Y \leftrightarrow Z$ is permitted since the absence of inducing paths between X, W and Y, Z, respectively, is guaranteed in any resulting graph.

Remark. Beyond Paper III, we have observed that there are more ways to prune local c-component scores than what was presented here. Most interestingly, unlike in BNSL, here we could even prune a local c-component based on c-components that contain *additional* parents. That is, pruning in MAGSL can go in both directions. For example, consider the local c-components consisting of nodes X, Y, Z in Figure 3.5. Since neither (b) nor (c) have external parents (incoming edges from other components), their extra internal parents cannot form (almost) directed paths which would violate ancestrality or maximality in a surrounding mixed graph. Thus we can always safely replace component (a) in a MAG with either (b) or (c). Therefore the "... is a subgraph of ..." conditions in our pruning rules could be replaced with a condition stating that the component(s) can contain additional edges as long as those edges satisfy certain requirements which prevent non-ancestrality and non-maximality. These refinements are not considered in the empirical results presented in Paper III or in the following experiments.

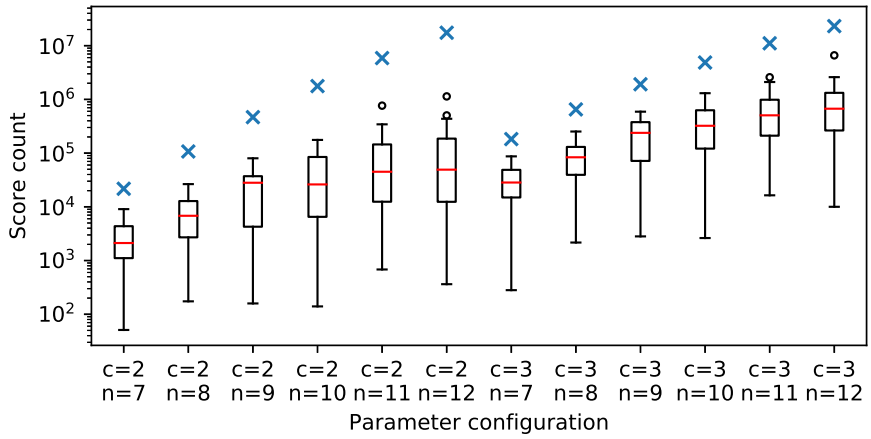


Figure 3.6: Score counts for the synthetic datasets.

3.3 Empirical Evaluation

In this section we empirically evaluate the performance of the proposed methods. We implemented the local score computation and exact search in C++ and both implementations are available online in open source. The score computation software includes an efficient implementation of the RICF algorithm and score pruning. For the evaluation we generated synthetic datasets with 200 samples each by sampling MAGs over $n = 7 \ldots 12$ nodes with expected degree d of 2 or 3.

In order to make score computation feasible, we use parent set limits $p = 8$ and $p = 4$ for c-component limits $c = 2$ and $c = 3$, respectively. With these configurations computing (and pruning) local scores for a 12-variable dataset can be done in around 10 hours.

Score Computation and Pruning. Figure 3.6 shows the number of local c-component scores produced for different number of nodes n and c-component size limits c on the synthetic data. The blue markers represent the number of scores without pruning (constant for each n, c configuration) whereas the box plots show empirically the number of scores after pruning.

We can see that the number of scores left after pruning considerably varies between different datasets. The figure also shows that increasing the c-component limit from 2 to 3 significantly increases the number of local scores, more so than increasing the number of nodes.

Running Time Performance. Figure 3.7 shows the running times (sorted in increasing order per run time for each line) of our exact search under the c-

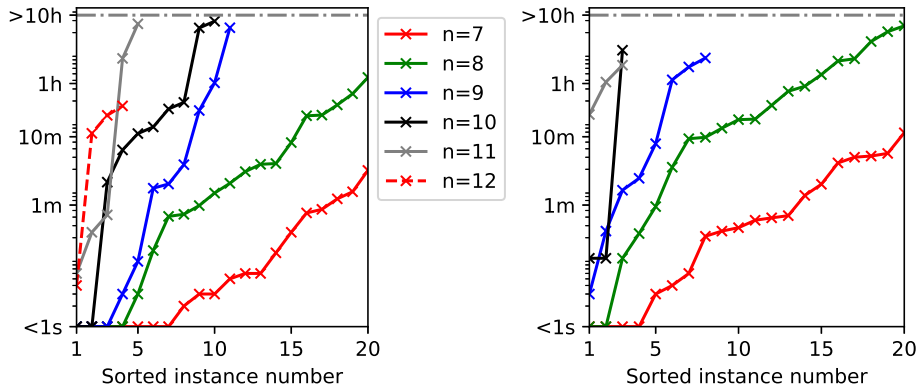


Figure 3.7: The running time performance of our exact search for $c = 2$ (left) and $c = 3$ (right) on synthetic data.

component size limits $c = 2$ and $c = 3$ on the synthetic data. Within approximately one hour, the approach scales up to 8-variable instances for $c = 2, 3$. For $c = 2$ we can solve many instances with 9-12 nodes within the per-instance time limit of 10 hours.

Quality of Solutions. We compare the quality of the solutions found by our exact search to those found by M3HC [92], FCI [80, 100] and GFCI [56]. As the other methods are considerably faster, we also bootstrapped the data 50 times and report the best result based on the score with respect to the original data.

Figure 3.8 shows the Bayes factors of the found MAGs with respect to baseline optimal Bayesian networks found by GOBNILP [3] on the synthetic data. The Bayes factors are based on the Gaussian BIC scores of the graphs found by each method. For scoring each local c-component, RICF was ran 100 times from different starting points and the highest found likelihood value was chosen. We used this strategy for computing the Gaussian BIC scores for the found MAGs from each method as well as for producing the local scores for our exact search. We can observe that there is a noticeable variation among datasets as to how much better the found MAGs are compared to the optimal BNs with respect to the Bayes factors. For some datasets BNs are as good as MAGs but in many cases MAGs are ten times better than BNs. FCI and GFCI appear to be unable to find high-scoring MAGs over the 200 samples in the data, whereas the bootstrapped M3HC often finds MAGs which are better than BNs. Our exact approach is still more accurate, finding MAGs that are usually multiple times better than the optimal BNs with respect to the Bayes factors.

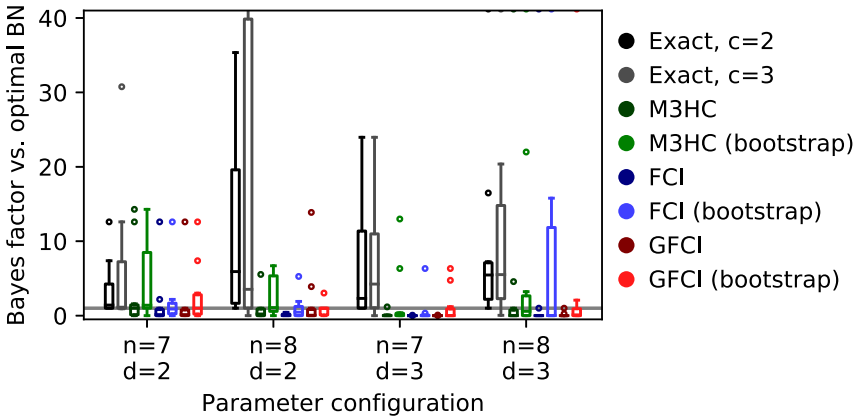


Figure 3.8: Bayes factors against optimal Bayesian networks on synthetic data from MAGs with average node degree d. Our exact search was tested with maximum c-component sizes $c = 2$ and $c = 3$.

Finally, we turn our focus to comparing the methods on four Gaussian Bayesian network benchmark datasets, namely arth150, ecoli70, magic-irri and magic-niab, from bnlearn repository [75]. We sampled from $N = 100$ to $N = 1600$ samples from each benchmark, and marginalized the data to n highly correlated variables in order to study structure learning in the presence of latent variables.

Table 3.1 compares the MAGs found by our exact search with $c = 3$ and $c = 4$ to the MAGs found by the in-exact methods on the benchmark data. Both $c = 3$ and $c = 4$ used parent set limit $p = 4$. The table is condensed to fit the page; here the "FCI", "GFCI" and "M3HC" columns contain the best BIC score among the MAG found for the original data *and* the MAGs found when the data was bootstrapped 50 times. Paper III contains more detailed results and a separate larger table containing results for $c = 2$. Table 3.1 shows that FCI, GFCI and M3HC can find high-scoring MAGs in some cases, especially for arth150 and ecoli70. For magic-niab and magic-irri our exact search obtains clearly better solutions on this metric than the in-exact methods. Moreover, the solutions found by the exact search are optimal for the input local scores with the c and p limits.

Table 3.1: Results on the benchmark datasets. The highest scores for each row are in bold. The full table, including running times of the exact search and the BIC scores of the optimal BNs, is found in Paper III.

			Gaussian BIC Score				
Dataset	N	n	**Exact** $c = 3$	**Exact** $c = 4$	**FCI**	**GFCI**	**M3HC**
arth150	100	7	-224.74	-224.74	**-219.27**	-227.01	-305.74
arth150	200	7	-377.96	-377.96	**-358.42**	-379.52	-418.23
arth150	400	7	-722.47	-722.47	**-661.35**	-727.62	-731.87
arth150	800	7	-1380.36	-1380.36	**-1235.14**	-1389.68	-1389.73
arth150	1600	7	-2696.91	-2696.91	**-2500.61**	-2724.3	-2703.04
ecoli70	100	7	-602.8	-602.8	-571.23	**-540.04**	-602.8
ecoli70	200	7	-1180.74	-1180.74	-1181.78	**-1055.2**	-1180.74
ecoli70	400	7	**-2321.64**	**-2321.64**	**-2321.64**	**-2321.64**	**-2321.64**
ecoli70	800	7	**-4614.11**	**-4614.11**	**-4614.11**	**-4614.11**	**-4614.11**
ecoli70	1600	7	**-9326.18**	**-9326.18**	**-9326.18**	**-9326.18**	**-9326.18**
magic-irri	100	7	**-977.01**	**-977.01**	-978.75	**-977.01**	-978.75
magic-irri	200	7	**-1951.54**	**-1951.54**	-1955.01	-1952.94	-1953.68
magic-irri	400	7	**-3802.84**	**-3802.84**	-3804.46	-3803.83	-3804.46
magic-irri	800	7	**-7520.48**	**-7520.48**	-7525.01	**-7520.48**	**-7520.48**
magic-irri	1600	7	**-15051.99**	**-15051.99**	-15054.51	-15052.84	**-15051.99**
magic-irri	100	8	-1091.59	*timeout*	**-1085.44**	-1092.7	-1094.05
magic-irri	200	8	**-2169.11**	*timeout*	-2174.51	-2171.45	-2169.22
magic-irri	400	8	**-4228.14**	*timeout*	-4230.37	-4229.09	-4230.23
magic-irri	800	8	**-8358.48**	*timeout*	-8363.27	-8360.05	**-8358.48**
magic-irri	1600	8	*timeout*	*timeout*	-16726.55	**-16718.89**	-16719.3
magic-niab	100	7	**-845.77**	**-845.77**	-846.58	**-845.77**	-856.6
magic-niab	200	7	-1616.51	**-1616.27**	-1617.77	-1617.4	-1618.87
magic-niab	400	7	**-3593.07**	**-3593.07**	-3593.21	-3593.17	-3594.87
magic-niab	800	7	**-6618.23**	**-6618.23**	-6621.76	-6618.42	-6620.72
magic-niab	1600	7	**-13206.57**	**-13206.57**	-13209.71	-13206.65	-13214.8
magic-niab	100	8	**-974.23**	**-974.23**	-976.04	**-974.23**	-985.05
magic-niab	200	8	**-1850.11**	**-1850.11**	-1853.83	-1851.43	-1856.23
magic-niab	400	8	**-4109.82**	*timeout*	-4113.02	-4110.93	-4113.53
magic-niab	800	8	**-8776.99**	*timeout*	-8780.14	-8777.64	-8779.73
magic-niab	1600	8	**-15306.12**	*timeout*	-15324.25	-15308.73	-15311.86

Chapter 4

Learning Causal Graphs

The MAG structures in Chapter 3 were a natural generalization of Bayesian network structures taking latent confounders into consideration. However, the ancestrality and maximality requirements in MAGs still limit what kind of causal structures can be represented with them. In particular, MAGs (and hence also Bayesian networks and chordal Markov networks) are unable to express cyclic causal relationships [1, 26, 79]. These relationships can occur for example in biological systems when empirical measurements take place at a slower rate than the causally relevant time steps of the measured objects [32].

As a classic example of cyclicity, consider the relationship between supply and demand [32]. Given a time series with sufficient time resolution, demand would always precede supply and thus the true structure between the two could be represented with an acyclic model. However, in practice both supply and demand have an effect on each other and the interactions between the two can occur in a span of days or weeks. In fact, these interactions can happen at such pace that measurements of supply and demand are unable to fully keep up with them. Acyclic models are not adequate for representing such situations and hence we need cyclic causal models [27, 33, 68].

This chapter overviews Papers IV and V, presenting an exact constraint-based branch-and-bound algorithm for learning causal graphs with cycles and latent confounders. Section 4.1 provides an overview of causal graphs and the related optimization problem. In Section 4.2 we explain how our algorithm solves the problem, and Section 4.3 proposes ways to further extend the search. Finally, Section 4.4 discusses empirical results comparing the practical performance of our algorithm against competing methods.

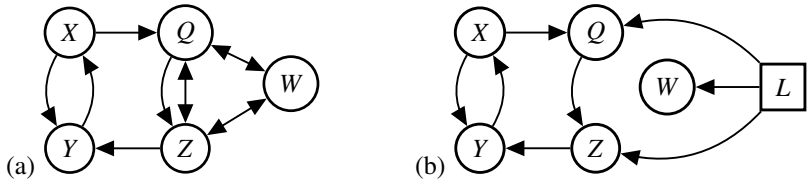


Figure 4.1: (a) Causal graph in the canonical representation using bidirected edges. (b) Causal graph with an unobserved variable L.

4.1 Causal Graphs

A *causal graph*, in the context of this thesis, is a mixed graph $G = (V, E)$ containing a combination of directed $X \to Y$ and bidirected $X \leftrightarrow Y$ edges. Unlike maximal ancestral graphs, causal graphs do not have any maximality or ancestrality requirements and hence any combination of edges yields a feasible structure. This enables them to feature directed cycles for representing feedback [66,68,79].

A causal graph is illustrated in Figure 4.1. As in the case of MAGs, the bidirected edges are used to represent latent confounders. This allows for a canonical representation of causal structures as a graph over the observed variables [61,80].

We consider learning causal graphs using constraint-based causal discovery where the idea is to obtain a set of independence and dependence constraints by running statistical independence tests on the data [10, 27, 36, 48, 98]. This yields a non-negative weight $w(k)$ for each constraint k.

Example 15. Suppose the data contains three nodes X, Y and Z. The (in)dependence constraints $\mathcal{K}$ could 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)

Given the weighted (in)dependence constraints, the weight of a causal graph is the total weight of the constraints not satisfied by the graph. The learning task is then to find a causal graph G^* with the minimum weight [33]. Formally, given set $\mathcal{K}$ of weighted (in)dependence constraints over variables V and a collection $\mathcal{G}$ of all causal graphs over V, the objective is to find a causal graph G^* such that

$$G^* \in \operatorname{argmin}_{G \in \mathcal{G}} \sum_{k \in \mathcal{K} \,:\, G \not\models k} w(k). \tag{4.1}$$

In order to determine whether an (in)dependence constraint is satisfied in a graph, we will use *d-separation* which is the central reachability criterion for causal

graphs [33, 61]. Under some well-motivated assumptions (detailed in Paper IV), two random variables are conditionally independent given a set of variables C (called a *conditioning set*) if and only if they are d-separated given C in the generating causal structure. We follow an equivalent definition of d-separation by Studený [84] which concerns the existence of certain types of walks in the graph, as explained next.

A *walk* in a graph 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. That is, Y is a collider if we have either $X \to Y \leftarrow Z$, $X \leftrightarrow Y \leftarrow Z$ or $X \to Y \leftrightarrow Z$, otherwise it is a non-collider.

Definition 4 (d-separation)**.** *Given a conditioning set C, a walk between $X, Y \notin C$ is* d-connecting *in a causal graph if every node Z in the walk (excluding X and Y) is either:*

(a) a collider satisfying $Z \in C$; or
(b) a non-collider satisfying $Z \notin C$.

Nodes are d-connected given C (denoted with $X \not\perp\!\!\!\perp Y \mid C$) if and only if there is a d-connecting walk between them. Otherwise they are d-separated (denoted with $X \perp\!\!\!\perp Y \mid C$).

Example 16. Consider Figure 4.1 (a). We have $X \not\perp\!\!\!\perp W \mid Y$ because of the following d-connecting walk: $X \to Q \to Z \to Y \leftarrow Z \leftrightarrow W$. We visited Z twice in order to make it a non-collider. We have $Y \perp\!\!\!\perp W \mid X, Z$ because either X or Z is a non-collider in all possible walks between Y and W.

4.2 Branch and Bound for Learning Causal Graphs

This section overviews our exact branch-and-bound algorithm for learning optimal causal graphs as proposed in Paper IV. We start by providing an overview of the search in Subsection 4.2.1. Subsection 4.2.2 explains how the satisfiability of the (in)dependence constraints is determined during the search, and Subsection 4.2.3 shows how this satisfiability information can be used to compute lower bounds using unsatisfiable core patterns.

4.2.1 Overview

The branch-and-bound algorithms we proposed in the earlier chapters used partial solutions which were extended by adding new nodes. In contrast, here we consider partial solutions (causal graphs) which already contain all the relevant nodes of the

optimization problem and are extended by deciding the edges between the nodes. That is, any edge can either be decided *present*, decided *absent* or be *undecided*. Figure 4.2 (a) shows an example.

Our depth-first search works as follows. We start off with a partial solution in which every possible edge between the nodes is undecided. Then, we start deciding the edges of the partial solution one by one. That is, at every point in the search tree we open up two new branches: one where some undecided edge is decided absent and one where it is decided present. Whenever all the edges have been decided, we have obtained a feasible solution candidate for the problem.

The algorithm keeps track of the lowest-weight solution candidate G^* found so far in the search. We can backtrack in the search whenever (a) all the edges of the current partial solution are decided or (b) the lower bound of the current partial solution (Subsection 4.2.3) is not lower than the weight of G^*. Once there are no more branches and partial solutions to visit, the search has finished and G^* is the optimal solution.

Branching heuristics. At each point in the search we have to choose the next undecided edge to branch with. To do this, we compare the endpoints (pairs of nodes) of each edge candidate using the following principles:

- Favor node pairs which share more undecided edges in the partial solution compared to the other node pairs. This allows the search to decide edges evenly throughout the graph instead of deciding all the edges between a single pair of nodes immediately.
- Favor node pairs where the corresponding variables appear to be more independent compared to other variable pairs. For measuring how independent variables appear, we compare the total weight of the *undetermined independence constraints* as detailed in Subsection 4.2.2. This is somewhat similar to how the PC algorithm [80] greedily decides an edge absent between a pair of nodes if an independence is found between the corresponding variables.

When a pair of nodes and an undecided edge between them has been chosen, we first visit the branch where the edge is decided absent. A more detailed explanation of the branching heuristics can be found in Paper IV.

4.2.2 Evaluating Constraints

As mentioned in Subsection 4.2.1, the weight of a causal graph is the sum of the weights of the unsatisfied (in)dependence constraints in the graph. Conversely, computing the exact weight is not possible for partial solutions due to the unde-

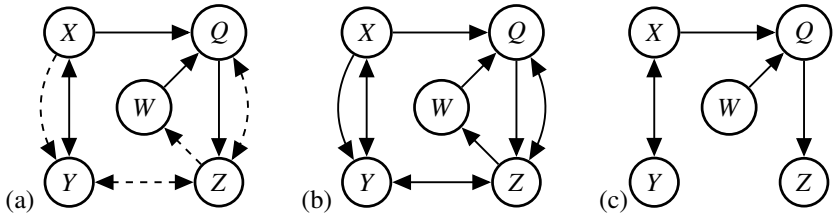


Figure 4.2: (a) Partial solution G and its completions (b) $\textsc{maxc}(G)$ and (c) $\textsc{minc}(G)$. Solid edges have been decided present, edges not shown have been decided absent and dashed edges represent undecided edges.

cided edges. However, we can determine which (in)dependence constraints are already guaranteed to be unsatisfied based on the decided edges. For example, if the current partial solution has decided an edge $X \to Y$ present, then all independence constraints between X and Y are unsatisfied in every solution in the current search tree branch. We can use this to compute lower bounds in the search.

We define the *maximal completion* $\textsc{maxc}(G)$ of a partial solution G to be a version of G in which all the undecided edges are decided present, and the *minimal completion* $\textsc{minc}(G)$ of G to be a version of G in which all the undecided edges are decided absent. We call an independence constraint *determined* for G if it is either satisfied in $\textsc{maxc}(G)$ or unsatisfied in $\textsc{minc}(G)$. Similarly a dependence constraint is determined if it is either satisfied in $\textsc{minc}(G)$ or unsatisfied in $\textsc{maxc}(G)$. All other (in)dependence constraints are *undetermined*.

Example 17. Consider the partial solution in Figure 4.2. The independence constraint $Y \perp\!\!\!\perp W \mid Q$ is determined unsatisfied because the d-connecting walk $Y \leftrightarrow X \to Q \leftarrow W$ exists regardless of how the undecided edges are decided. Meanwhile the dependence constraint $Y \not\perp\!\!\!\perp Z \mid X$ is still undetermined because its satisfiability depends on whether the edge $Y \leftrightarrow Z$ is decided present.

4.2.3 Core-based Lower Bounds

A naive lower bound for the current partial solution is obtained by summing up the weights of the determined unsatisfied contraints. To obtain tighter lower bounds we make use of so-called *unsatisfiable core patterns* by Hyttinen *et al.* [35].

In this context an unsatisfiable core is a set of (in)dependence constraints which cannot all be simultaneously satisfied by any causal graph. For example the following three constraints are proven to form an unsatisfiable core for any variables X, Y, Z and a conditioning set C [35]:

$$\{X \perp\!\!\!\perp Z \mid C; \quad X \perp\!\!\!\perp Y \mid C; \quad X \not\perp\!\!\!\perp Z \mid Y, C\}.$$

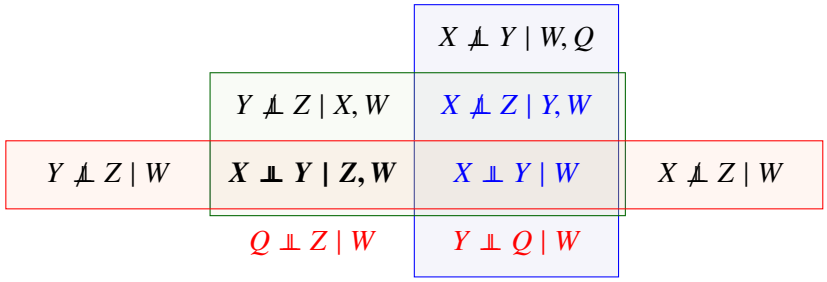


Figure 4.3: An example of core-based lower bounding.

Consequently, if the constraints $X \perp\!\!\!\perp Z \mid C$ and $X \perp\!\!\!\perp Y \mid C$ have been determined satisfied in the current branch, then we know for certain that the remaining constraint $X \not\perp\!\!\!\perp Z \mid Y, C$ will be unsatisfied in all the solutions in the branch.

Similarly to Hyttinen *et al.* [35], we compute lower bounds by solving the following *minimum-weight hitting set problem* [17,51,73]: given a collection of unsatisfiable cores, find a minimum-weight set of constraints $\mathcal{U}$ that contains at least one constraint from each core. Here $\mathcal{U}$ corresponds to an optimal set of unsatisfied constraints in accordance to the cores. We make the hitting set problem branch-specific by enforcing that $\mathcal{U}$ must contain all constraints which are determined unsatisfied in the current partial solution and $\mathcal{U}$ cannot contain any constraints which are determined satisfied. The weight of $\mathcal{U}$ is then the core-based lower bound for the current partial solution.

Example 18. Suppose we have the cores in Figure 4.3 and the partial solution satisfies $X \not\perp\!\!\!\perp Z \mid Y, W$ and $X \perp\!\!\!\perp Y \mid W$ (in blue), and violates $Y \perp\!\!\!\perp Q \mid W$ and $Q \perp\!\!\!\perp Z \mid W$ (in red). One constraint in each core marked by the rectangles must be chosen. The violated constraint $Y \perp\!\!\!\perp Q \mid W$ hits the core marked by the vertical blue rectangle. If for simplicity the weights are all constants, the remaining cores can be optimally hit by $X \perp\!\!\!\perp Y \mid Z, W$ (in bold). Thus, the final lower bound for the partial solution is $w(Y \perp\!\!\!\perp Q \mid W) + w(Q \perp\!\!\!\perp Z \mid W) + w(X \perp\!\!\!\perp Y \mid Z, W)$, where the last term in the sum tightens the bound compared to the simple bound due to just violated constraints.

We gather unsatisfiable cores based on the given (in)dependence constraints in the beginning of the search using seven core patterns from [35]. In order to make the bound computations computationally feasible, we settle with solving the hitting set problem approximately via a linear programming relaxation. This may make the bounds slightly looser compared to finding an optimal hitting set but the exactness of the overall search is not affected.

4.2.4 Fast Constraint Evaluation

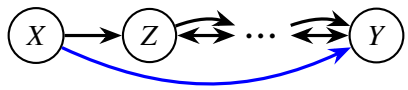
As explained in Subsection 4.2.2, we have to evaluate for every partial solution the states (undetermined, satisfied, unsatisfied) of the yet-undetermined constraints in the minimal and maximal completions in order to keep the lower bounds up to date. This can be computationally expensive due to the fact that there are super-polynomial number of (in)dependence constraints with respect to the number of nodes in the partial solution. Hence it would be useful to have other ways to check the satisfiability of constraints than individual d-separation tests. Fortunately there are several different ways for checking whether (in)dependence constraints are satisfied by a graph [76, 84]. Building on such ideas, we propose the following notions of *unavoidable colliders* and *unavoidable non-colliders*.

The goal is to identify bottlenecks between nodes in a graph which provably limit the connectivity of the corresponding variables. For example, if all walks between nodes X and Y contain some node Z as a non-collider (called an unavoidable non-collider), then we know that the graph satisfies every $X \perp\!\!\!\perp Y \mid C$ where $Z \in C$. Similarly, if Z is a collider (called an unavoidable collider) on all the walks, then the graph satisfies every $X \perp\!\!\!\perp Y \mid C'$ where $Z \notin C'$. Identifying some simple (un)avoidable (non)colliders can be done in polynomial time and is explained in Paper IV.

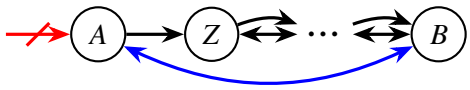
The downsides of unavoidable (non)colliders is that they only exist in sparse subgraphs and that they only reduce some of the d-separation tests between a pair of variables. For this reason we introduce even more powerful way to reduce the number of d-separation tests called *edge irrelevancy rules*. The rules aim to detect when the latest edge decision in a partial solution does not affect the satisfiability of *any* constraints between a pair of variables. The correctness of the rules are proven in Paper IV.

We use $V_1 \rightsquigarrow V_n$ in the rules to denote that a completion of a partial solution has edges $V_1 \to V_2 \to \ldots \to V_n$ and also $V_{i-1} \leftrightarrow V_1$ for all $i > 2$. Note that $X \rightsquigarrow Y$ describes a certain type of inducing path (recall p. 28) between X and Y and hence all dependence constraints are satisfied and all independence constraints are unsatisfied between the variables. In the simplest case $X \to Y$ implies $X \rightsquigarrow Y$.

Edge Irrelevancy Rule 1. Suppose setting $X \to Y$ present is the only edge decision made since the last time the constraints were evaluated. Then there is no need to evaluate constraints for any variables if $X \rightsquigarrow Y$.

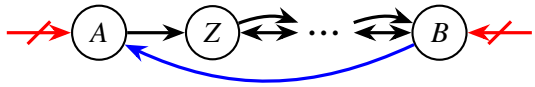


Edge Irrelevancy Rule 2. Suppose setting $A \leftrightarrow B$ present is the only edge decision made since the last time the constraints were evaluated. Then there is no need to evaluate constraints between variables X and Y if $A \rightsquigarrow B$ and either $X = A$, $Y = B$ or A does not have incoming edges besides from B.



Edge Irrelevancy Rule 3. Suppose setting $A \leftarrow B$ present is the only edge decision made since the last time the constraints were evaluated. Then there is no need to evaluate constraints between variables X and Y if $A \rightsquigarrow B$ and the following conditions hold:

- $A = X$ or $A = Y$ or A does not have incoming edges besides from B,
- $B = X$ or $B = Y$ or B does not have incoming edges besides from A.



The fourth and final rule is perhaps the most useful in practice as it involves deciding edges absent, aligning well with our branching heuristic which favors edge decisions between variables which appear to be independent.

Edge Irrelevancy Rule 4. Suppose setting some edge absent is the only edge decision made since the last time the constraints were evaluated. Then there is no need to evaluate constraints between variables X and Y if we still have either $X \rightsquigarrow Y$, $Y \rightsquigarrow X$ or $X \leftrightarrow Y$.

4.3 Extending the Algorithm

This section overviews two important extensions to our branch and bound as proposed in Paper V. Subsection 4.3.1 explains how the algorithm can be extended to support interventional data, and Subsection 4.3.2 shows how the search can use σ-separation as the reachability criterion instead of the classic d-separation.

4.3.1 Interventional Data

Sometimes we may be dealing with *interventional data* with multiple experimental datasets [48]. In that case independence constraints can take the form

$X \perp\!\!\!\perp Y \mid C \parallel J$ where J is called an *intervention set*. The satisfiability of such constraint in a graph can be checked simply by testing whether $X \perp\!\!\!\perp Y \mid C$ is satisfied after *intervening* on J, i.e., after we have removed all incoming edges to the nodes in J from the graph. Similarly $X \not\perp\!\!\!\perp Y \mid C \parallel J$ is equivalent to $X \not\perp\!\!\!\perp Y \mid C$ after intervening on J.

The unsatisfiable core patterns by Hyttinen *et al.* [35] are sound for interventional data with but they do not take advantage of the existence of multiple experimental datasets. For example $\{X \perp\!\!\!\perp Y;\ \ X \not\perp\!\!\!\perp Y \parallel J\}$ would be a valid unsatisfiable core in case of interventional data: if X and Y are d-separated, they cannot become d-connected by removing edges to J, and if X and Y are d-connected, they cannot become d-separated by adding edges to J.

Generalizing the above intuition, we can tighten the lower bounds of Subsection 4.2.3 by forming the following rules *across* different datasets. For every variable X, Y, conditioning set C and intervention sets $J \subset J'$:

$$X \perp\!\!\!\perp Y \mid C \parallel J \quad \Rightarrow \quad X \perp\!\!\!\perp Y \mid C \parallel J', \tag{4.2}$$

$$X \not\perp\!\!\!\perp Y \mid C \parallel J' \quad \Rightarrow \quad X \not\perp\!\!\!\perp Y \mid C \parallel J. \tag{4.3}$$

These rules are integrated into our linear programming relaxation to restrict which constraints the hitting set solution $\mathcal{U}$ can contain. For example we can only have constraint $X \not\perp\!\!\!\perp Y \mid C \parallel J$ in $\mathcal{U}$ if we also have either $X \not\perp\!\!\!\perp Y \mid C \parallel J' \in \mathcal{U}$ or equivalently $X \perp\!\!\!\perp Y \mid C \parallel J' \notin \mathcal{U}$.

4.3.2 Support for σ-separation

So far we have used d-separation as the reachability criterion for evaluating the satisfiability of the constraints. However, in case of non-linear relations (discrete variables) and cycles, d-separated variables may actually be dependent [54, 79]. To this end, Forré *et al.* [27] proposed the notion of σ-separation which is sound for non-linear relations and cycles under global compatibility of the local causal mechanisms.

Interestingly, σ-connection works very similarly to d-connection, but with the following exception. Suppose Z is a node in the conditioning set. In a d-connecting walk Z has to be a collider (case (a) in Definition 4), but in a σ-connecting walk Z can be either a collider or a non-collider. However, in order for Z to be a non-collider, the walk must contain a triple of adjacent nodes (X, Z, Y) such that:

1. if $X \to Z$, then the graph has a directed path from Z to X, and
2. if $Y \to Z$, then the graph has a directed path from Z to Y.

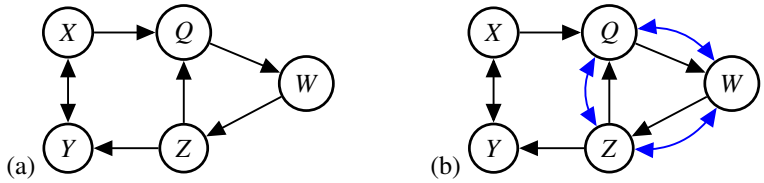


Figure 4.4: (a) Causal graph G, and (b) the σ-extension of G.

Example 19. Consider Figure 4.4 (a). We can see that the graph does not contain any d-connecting walks between Y and W given Q, Z. However, the graph does contain the following σ-connecting walk between Y and W given Q, Z: $Y \leftrightarrow X \rightarrow Q \rightarrow W$. Here Q is allowed to be a non-collider since the walk contains $Q \rightarrow W$ and the graph has a directed path from W to Q.

We propose the following notion of *σ-extension*. It transforms a causal graph G into another causal graph G' by adding bidirected edges so that d-connected variables in G' have the corresponding σ-connection in G. The proof can be found in Paper V.

Definition 5 (σ-extension)**.** *Let G be a causal graph. The σ-extension of G, G' contains all the edges in G, and an additional edge $X \leftrightarrow Y$ for any adjacent nodes X and Y that are in a same directed cycle.*

Example 20. In Figure 4.4 (b) we can see the σ-extension of graph (a). Since there is a directed cycle consisting of nodes Q, W, Z, the σ-extension contains additional edges $Q \leftrightarrow W$, $W \leftrightarrow Z$ and $Z \leftrightarrow Q$. As the result, graph (b) has a d-connecting walk between Y and W given Q, Z.

We integrate σ-separation into the search simply by evaluating the constraint states for the current partial solution G using d-separation tests in the σ-extensions of $\textsc{maxc}(G)$ and $\textsc{minc}(G)$. In addition, Paper V shows that the core-based lower bounds from Subsection 4.2.3 as well as the rules from Subsection 4.3.1 are valid in the case of σ-separation.

4.4 Empirical Evaluation

In this section we evaluate the practical performance of our branch and bound for learning optimal causal graphs in the generalized search space containing both cycles and latent confounders. Papers IV and V provide details about how the (in)dependence constraints were obtained for the algorithms in the following experiments. We implemented our branch and bound in C++ and will refer to it as *bcause*. The implementation is available online in open source.

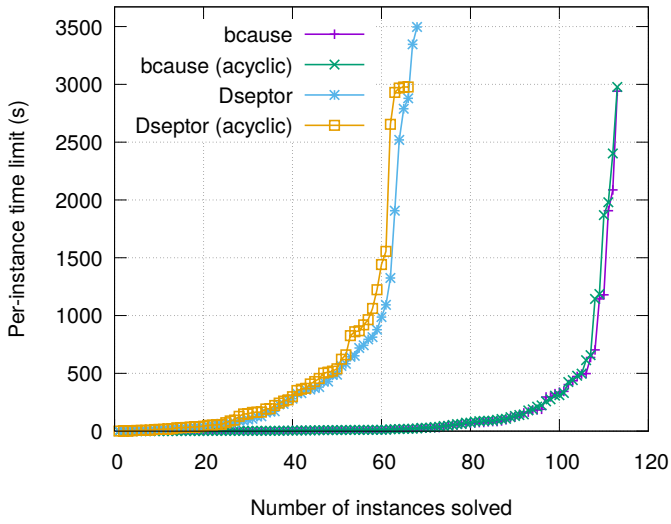


Figure 4.5: Comparison of bcause and Dseptor both in the case of an unrestricted search space and as well as in a search space restricted to acyclic graphs. The plot shows the number of instances solved (x-axis) for different per-instance time limits (y-axis).

Comparison with Dseptor. We begin by comparing the running time performance of bcause to that of Dseptor [35] which is the state-of-the-art approach for learning causal graphs in the generalized search space. For the comparison we use a number of real-world problem instances previously used for benchmarking causal discovery approaches [35]. The instances contain no experimental datasets and have 6 to 12 variables each. Here we consider the classic d-separation as the separation criterion as Dseptor does not currently support σ-separation.

Figure 4.5 shows that bcause is able to learn optimal causal graphs for more instances within a 1-hour timit limit compared to Dseptor. We can also observe that whether or not we allow cycles in the solutions does not significantly affect the running time of the algorithms. One explanation for this is that for many of the instances there exists an optimal solution which is acyclic.

Significance of the Marginal Contributions. Figure 4.6 shows the significance of two domain-specific techniques implemented in bcause: the marker types denote the number of variables in each instance. The impact of the core-based lower bounds of Subsection 4.2.2 can be seen in Figure 4.6 (left): the running time performance of the search is considerably improved compared to using naive bounds based solely on the determined unsatisfied constraints. Only for few instances the naive lower bounds yield a better solve time, mainly because of the computa-

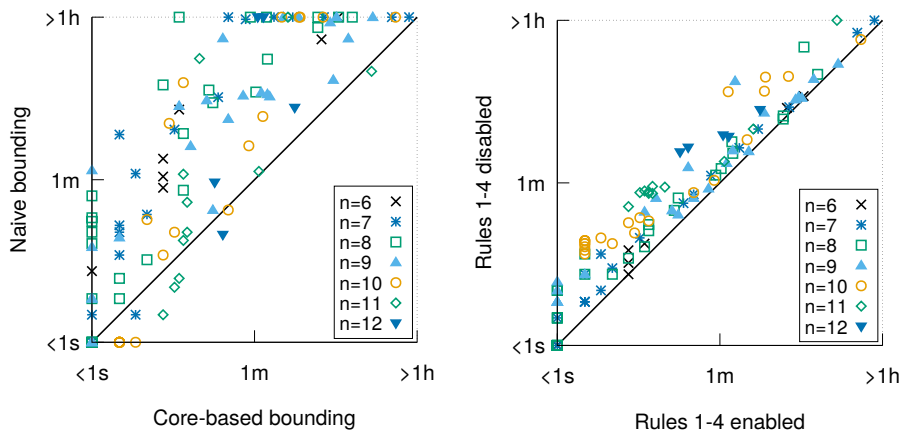


Figure 4.6: The effect of the marginal contributions on the running performance of our branch and bound. *Left:* core-based lower bounds vs naive lower bounds. *Right:* the four edge irrelevancy rules.

tional cost of solving the minimum-weight hitting set problem. Figure 4.6 (right) shows how the edge irrelevancy rules of Subsection 4.2.4 provide a consistent albeit lesser speed-up in the search.

Interventional Data and σ-Separation. We also evaluate the performance of our branch and bound in the case of interventional data featuring multiple experimental datasets and using σ-separation as the separation criterion. We will refer to our search as bcause+ when it is extended to this setting as detailed in Subsections 4.3.1 and 4.3.2.

We simulated data from non-linear cyclic causal models similarly to Forré and Mooij [27]. For comparison we consider the only two exact causal discovery methods that support σ-separation. First, the answer set programming (ASP) encoding by Forré and Mooij [27], and also an improved ASP encoding proposed in Paper V which we will refer to as "ASP+".

Figure 4.7 (left) shows a running time comparison between bcause+, ASP by Forré and Mooij [27] and ASP+. The instances contain 6 variables each and feature either one, three or five experimental datasets. We can observe from the figure that ASP+ consistently outperforms the other approaches, being able to learn optimal causal graphs for more instances within a 1-hour time limit. Meanwhile bcause+ does not seem to generalize particularly well to this setting. On the other hand, the running times of the ASP approaches take larger hit compared to bcause+ when the number of experimental datasets increases.

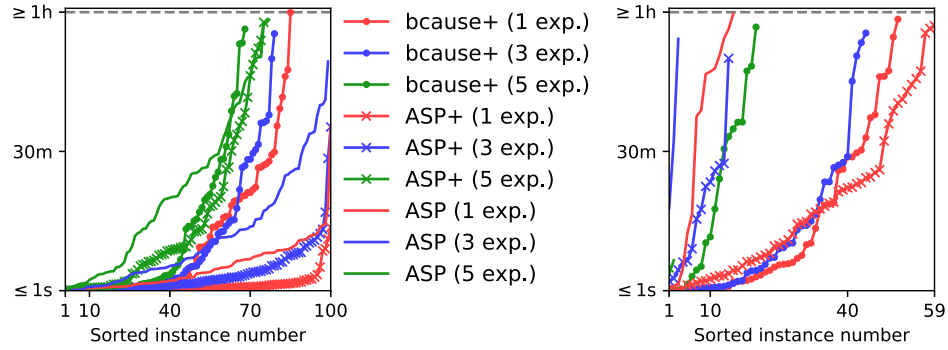


Figure 4.7: The running times of bcause+, ASP by Forré and Mooij [27] and ASP+ on instances with one ("1 exp."), three ("3 exp.") and five ("5 exp.") experimental datasets. *Left*: 6-node graphs. *Right*: 7-node graphs under joint causal sufficiency assumption.

Finally, we perform a comparison on 7-variable instances. Since all three approaches struggle to solve instances in this case, we also assume joint causal sufficiency, i.e., no bidirected edges in the learned graphs. The results are shown in Figure 4.7 (right). Here the ability of bcause+ to handle multiple experimental datasets becomes more apparent as it starts to outperform the ASP encodings in case of three or more experimental datasets.

Chapter 5

Conclusions

This thesis presented advances in structure learning for three classes of graphical models. A summary of the key findings is provided below.

For chordal Markov network structure learning (CMSL) we proposed two different algorithms. The first one is an exact branch-and-bound approach for learning optimal structures. The algorithm uses Bayesian networks as upper bounds and employs symmetry breaking and on-the-fly score pruning to reduce the size of the search space. The second proposed algorithm is a stochastic local search which is based on moving through the search space via three different neighborhood types and using on-the-fly score computation strategy to reduce the amount of scores needed to be computed for the search.

Both of our algorithms for CMSL have their own strengths and weaknesses. The exact branch-and-bound is able to find provably optimal networks containing up to 20 variables without restricting the search space, but on the other hand it suffers from timeouts even for smaller problem instances. While our stochastic local search does not give guarantees for the quality of its solutions, we observed empirically that it can often find optimal structures much faster than the state-of-the-art exact approaches, in addition to scaling considerably further. We also showed that all three proposed neighborhood types are crucial for the performance of the local search.

For maximal ancestral graph structure learning (MAGSL) we presented the first algorithm that is exact with respect to given local scores. The approach uses dynamic programming for learning optimal ancestral graphs (AGs) and branch and bound for finding the highest-scoring maximal solution among the AGs. We also developed a method for generating and pruning local scores for MAGSL in order to make exact search feasible. Empirically we observed that pruning

significantly reduces the number of local scores. We also showed how our exact search is often able to achieve higher accuracy (find better structures) than the existing in-exact approaches for MAGSL.

For learning causal graphs with cycles and latent confounders we introduced a constraint-based branch-and-bound approach. The algorithm computes upper bounds by solving the minimum-weight hitting set problem over unsatisfiable cores via a linear programming relaxation. To make constraint evaluation efficient, the branch and bound employs ways of detecting when changes to a partial solutions cannot affect the satisfiability of certain (in)dependence constraints. We further extended the algorithm to support interventional data and σ-separation as the graphical separation criterion.

Empirically we showed that our branch and bound finds optimal causal graphs faster than a recent exact algorithm for the problem. We also evaluated the branch and bound on interventional data and by using σ-separation as the separation criterion, and observed that the search is able to handle higher number of experimental datasets than the only causal discovery approaches supporting σ-separation.

There are still many interesting discoveries to be made when it comes to learning structures for the model classes studied in this thesis. Thus we end with a discussion about promising ideas for further work.

Unlike in the case of CMSL, our MAGSL approach does not currently make use of any kind of symmetry breaking rules, meaning that it may unnecessarily iterate multiple solutions belonging to a same equivalence class. Breaking these symmetries could substantially speed up the search, especially considering how much larger the space of MAGs is compared to the space of chordal Markov networks.

The success of our stochastic local search for CMSL raises the question whether a local search algorithm would also be useful for learning causal graphs in the presence cycles and latent confounders. After all, the exact approaches for the problem currently only scale up to around 12 variables. A stochastic local search could also evaluate the weights of its solution candidates approximately, thus circumventing the need to constantly deal with a superpolynomial number of (in)dependence constraints (with respect to the number of variables) like in the case of the exact methods. Discovering a good polynomial-time method for approximating the weight of a causal graph certainly presents an interesting challenge.

Another promising direction for future work would be to explore how our branch and bound for learning causal graphs could take more effectively advantage of σ-separation as the reachability criterion. Currently everything in the search from heuristics to constraint evaluation is still tailored for d-separation. Hence the algorithm would benefit from the development of σ-separation-specific search tech-

niques. As an example, the branch and bound could benefit from σ-separation in the following way: whenever a new cycle is formed in a partial solution, the search could backtrack if a bidirected edge had been decided absent between *any* adjacent nodes in the cycle. This is a direct consequence of the σ-extension from Subsection 4.3.2 and essentially represents a form of symmetry breaking, possibly significantly reducing the size of the search space.

There is still more to explore in terms of score pruning techniques for MAGSL as mentioned at the end of Subsection 3.2.3. Since local c-components are a generalization of the much simpler BN local scores, they also allow for more complex pruning techniques. Not only can components be pruned due to their subgraphs (like in BNSL), they can also be pruned due to their supergraphs, and even due to graphs which are neither. A promising direction for future work would be to form a more comprehensive set of pruning rules for MAGSL which could also speed up the exact score-based search.

References

[1] C. Améndola, P. Dettling, M. Drton, F. Onori, and J. Wu. Structure learning for cyclic linear causal models. In *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI 2020)*, volume 124 of *Proceedings of Machine Learning Research*, pages 999–1008. PMLR, 2020.

[2] F. R. Bach and M. I. Jordan. Thin junction trees. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pages 569–576. MIT Press, 2001.

[3] M. Bartlett and J. Cussens. Integer linear programming for the Bayesian network structure learning problem. *Artificial Intelligence*, 244:258–271, 2017.

[4] D. Bernstein, B. Saeed, C. Squires, and C. Uhler. Ordering-based causal structure learning in the presence of latent variables. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS 2020)*, Proceedings of Machine Learning Research, pages 4098–4108. PMLR, 2020.

[5] A. Chechetka and C. Guestrin. Efficient principled learning of thin junction trees. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, pages 273–280. Curran Associates, Inc., 2007.

[6] D. M. Chickering. A transformational characterization of equivalent Bayesian network structures. In *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI 1995)*, pages 87–98. Morgan Kaufmann, 1995.

[7] D. M. Chickering. Learning equivalence classes of Bayesian network structures. *Journal of Machine Learning Research*, 2:445–498, 2002.

[8] D. M. Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, 2002.

[9] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions of Information Theory*, 14(3):462–467, 1968.

[10] T. Claassen and T. Heskes. A Bayesian approach to constraint based causal inference. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI 2012)*, pages 207–216. AUAI Press, 2012.

[11] T. Claassen, J. M. Mooij, and T. Heskes. Learning sparse causal models is not NP-hard. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI 2013)*, pages 172–181. AUAI Press, 2013.

[12] D. Colombo and M. H. Maathuis. Order-independent constraint-based causal structure learning. *Journal of Machine Learning Research*, 15(1):3741–3782, 2014.

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

[14] J. Corander, T. Janhunen, J. Rintanen, H. J. Nyman, and J. Pensar. Learning chordal Markov networks by constraint satisfaction. In *Advances in Neural Information Processing Systems 26 (NIPS 2013)*, pages 1349–1357. Curran Associates, Inc., 2013.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[16] A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[17] J. Davies and F. Bacchus. Exploiting the power of MIP solvers in MAXSAT. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT 2013)*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013.

[18] P. Dawid and S. L. Lauritzen. Hyper Markov laws in the statistical analysis of decomposable graphical models. *Annals of Statistics*, 21(3):1272–1317, 1993.

[19] C. P. de Campos and Q. Ji. Properties of Bayesian Dirichlet scores to learn Bayesian network structures. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*. AAAI Press, 2010.

[20] C. P. de Campos and Q. Ji. Efficient structure learning of Bayesian networks using constraints. *Journal of Machine Learning Research*, 12:663–689, 2011.

[21] A. Deshpande, M. N. Garofalakis, and M. I. Jordan. Efficient stepwise selection in decomposable models. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI 2001)*, pages 128–135. Morgan Kaufmann, 2001.

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

[23] M. Drton and T. S. Richardson. Multimodality of the likelihood in the bivariate seemingly unrelated regressions model. *Biometrika*, 91(2):383–392, 06 2004.

[24] D. Edwards. *Introduction to Graphical Modelling*. Springer, 2000.

[25] F. V. Fomin and Y. Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM Journal on Computing*, 42(6):2197–2216, 2013.

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

[27] P. Forré and J. M. Mooij. Constraint-based causal discovery for non-linear structural causal models with cycles and latent confounders. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI 2018)*, pages 269–278. AUAI Press, 2018.

[28] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[29] P. Gillot and P. Parviainen. Scalable Bayesian network structure learning via maximum acyclic subgraph. In *Proceedings of the 10th International Conference on Probabilistic Graphical Models (PGM 2020)*, volume 138 of *Proceedings of Machine Learning Research*, pages 209–220. PMLR, 2020.

[30] V. Gogate, W. A. Webb, and P. M. Domingos. Learning efficient Markov networks. In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, pages 748–756. Curran Associates, Inc., 2010.

[31] H. Hoos and T. Stützle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., 2004.

[32] A. Hyttinen, F. Eberhardt, and P. O. Hoyer. Learning linear cyclic causal models with latent variables. *Journal of Machine Learning Research*, 13(109):3387–3439, 2012.

[33] A. Hyttinen, F. Eberhardt, and M. Järvisalo. Constraint-based causal discovery: Conflict resolution with answer set programming. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence (UAI 2014)*, pages 340–349. AUAI Press, 2014.

[34] A. Hyttinen, P. O. Hoyer, F. Eberhardt, and M. Järvisalo. Discovering cyclic causal models with latent variables: A general SAT-based procedure. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI 2013)*, pages 301–310. AUAI Press, 2013.

[35] A. Hyttinen, P. Saikko, and M. Järvisalo. A core-guided approach to learning optimal causal graphs. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, pages 645–651. ijcai.org, 2017.

[36] F. Jabbari, J. D. Ramsey, P. Spirtes, and G. F. Cooper. Discovery of causal models that contain latent variables through Bayesian scoring of independence constraints. In *Proceedings of the European Conference on Machine Learning & Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2017)*, volume 10535 of *Lecture Notes in Computer Science*, pages 142–157. Springer, 2017.

[37] A. Jaber, J. Zhang, and E. Bareinboim. On causal identification under Markov equivalence. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 6181–6185. ijcai.org, 2019.

[38] T. Janhunen, M. Gebser, J. Rintanen, H. J. Nyman, J. Pensar, and J. Corander. Learning discrete decomposable graphical models via constraint optimization. *Statistics and Computing*, 27(1):115–130, 2017.

[39] K. Kangas, M. Koivisto, and T. M. Niinimäki. Learning chordal Markov networks by dynamic programming. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pages 2357–2365. Curran Associates, Inc., 2014.

[40] K. Kangas, T. M. Niinimäki, and M. Koivisto. Averaging of decomposable graphs by dynamic programming and sampling. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence (UAI 2015)*, pages 415–424. AUAI Press, 2015.

[41] D. Koller and N. Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009.

[42] K. S. S. Kumar and F. R. Bach. Convex relaxations for learning bounded-treewidth decomposable graphs. In *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 525–533. JMLR.org, 2013.

[43] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988.

[44] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. In *Readings in Uncertain Reasoning*, pages 415–448. Morgan Kaufmann Publishers Inc., 1990.

[45] C. Lee and P. van Beek. An experimental analysis of anytime algorithms for Bayesian network structure learning. In *Proceedings of the 3rd Workshop on Advanced Methodologies for Bayesian Networks (AMBN 2017)*, volume 73 of *Proceedings of Machine Learning Research*, pages 69–80. PMLR, 2017.

[46] C. Lee and P. van Beek. Metaheuristics for score-and-search Bayesian network structure learning. In *Proceedings of the 30th Canadian Conference on Artificial Intelligence (Canadian AI 2017)*, volume 10233 of *Lecture Notes in Computer Science*, pages 129–141, 2017.

[47] G. Letac and H. Massam. Wishart distributions for decomposable graphs. *Annals of Statistics*, 35(3):1278–1323, 2007.

[48] S. Magliacane, T. Claassen, and J. M. Mooij. Ancestral causal inference. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, pages 4466–4474, 2016.

[49] F. M. Malvestuto. Approximating discrete probability distributions with decomposable models. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1287–1294, 1991.

[50] V. W. Marek and M. Truszczynski. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm - A 25-Year Perspective*, Artificial Intelligence, pages 375–398. Springer, 1999.

[51] E. Moreno-Centeno and R. M. Karp. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. *Operations Research*, 61(2):453–468, 2013.

[52] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[53] M. Narasimhan and J. A. Bilmes. PAC-learning bounded tree-width graphical models. In *Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence (UAI 2004)*, pages 410–417. AUAI Press, 2004.

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

[55] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 11 1999.

[56] J. M. Ogarrio, P. Spirtes, and J. Ramsey. A hybrid causal search algorithm for latent variable models. In *Proceedings of the 8th International Conference on Probabilistic Graphical Models (PGM 2016)*, volume 52 of *JMLR W&CP*, pages 368–379. JMLR.org, 2016.

[57] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., 1982.

[58] P. Parviainen and M. Koivisto. Bayesian structure discovery in Bayesian networks with less space. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9 of *JMLR Proceedings*, pages 589–596. JMLR.org, 2010.

[59] P. Parviainen and M. Koivisto. Finding optimal Bayesian networks using precedence constraints. *Journal of Machine Learning Research*, 14(1):1387–1415, 2013.

[60] J. Pearl. *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

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

[62] E. Perkovic, J. Textor, M. Kalisch, and M. H. Maathuis. Complete graphical characterization and construction of adjustment sets in Markov equivalence classes of ancestral graphs. *Journal of Machine Learning Research*, 18:220:1–220:62, 2017.

[63] S. D. Pietra, V. J. D. Pietra, and J. D. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

[64] J. Ramsey, J. Zhang, and P. Spirtes. Adjacency-faithfulness and conservative causal inference. *Computing Research Repository*, abs/1206.6843, 2012.

[65] K. Rantanen, A. Hyttinen, and M. Järvisalo. Learning optimal causal graphs with exact search. In *Proceedings of the 9th International Conference on Probabilistic Graphical Models (PGM 2018)*, volume 72 of *Proceedings of Machine Learning Research*, pages 344–355. PMLR, 2018.

[66] T. Richardson. A discovery algorithm for directed cyclic graphs. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI 1996)*, pages 454–461. Morgan Kaufmann, 1996.

[67] T. Richardson and P. Spirtes. Ancestral graph Markov models. *Annals of Statistics*, 30(4):962–1030, 2002.

[68] T. S. Richardson. *Feedback Models: Interpretation and Discovery*. PhD thesis, Carnegie Mellon University, 1996.

[69] T. S. Richardson. A factorization criterion for acyclic directed mixed graphs. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI 2009)*, pages 462–470. AUAI Press, 2009.

[70] T. S. Richardson and P. Spirtes. Causal inference via ancestral graph models. *Oxford Statistical Science Series*, 1(27):83–105, 2003.

[71] D. Rose, R. Tarjan, and G. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.

[72] P. Saikko, J. Berg, and M. Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT 2016)*, volume 9710 of *Lecture Notes in Computer Science*, pages 539–546. Springer, 2016.

[73] P. Saikko, J. P. Wallner, and M. Järvisalo. Implicit hitting set algorithms for reasoning beyond NP. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR 2016)*, pages 104–113. AAAI Press, 2016.

[74] M. Scanagatta, C. P. de Campos, G. Corani, and M. Zaffalon. Learning Bayesian networks with thousands of variables. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, pages 1864–1872, 2015.

[75] M. Scutari. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3):1–22, 2010.

[76] R. D. Shachter. Bayes-Ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI 1998)*, pages 480–487. Morgan Kaufmann, 1998.

[77] D. Shahaf and C. Guestrin. Learning thin junction trees via graph cuts. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, volume 5 of *JMLR W&CP*, pages 113–120. JMLR.org, 2009.

[78] T. Silander and P. Myllymäki. A simple approach for finding the globally optimal Bayesian network structure. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI 2006)*. AUAI Press, 2006.

[79] P. Spirtes. Directed cyclic graphical representations of feedback models. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI 1995)*, pages 491–498. Morgan Kaufmann, 1995.

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

[81] P. Spirtes and T. S. Richardson. A polynomial time algorithm for determining DAG equivalence in the presence of latent variables and selection bias. In *Proceedings of the 6th International Workshop on Artificial Intelligence and Statistics (AISTATS 1997)*, volume R1 of *Proceedings of Machine Learning Research*, pages 489–500. PMLR, 1997.

[82] N. Srebro. Maximum likelihood bounded tree-width Markov networks. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI 2001)*, pages 504–511. Morgan Kaufmann, 2001.

[83] N. Srebro. Maximum likelihood bounded tree-width Markov networks. *Artificial Intelligence*, 143(1):123–138, 2003.

[84] M. Studený. Bayesian networks from the point of view of chain graphs. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI 1998)*, 1998.

[85] M. Studený and J. Cussens. Towards using the chordal graph polytope in learning decomposable models. *International Journal of Approximate Reasoning*, 88:259–281, 2017.

[86] J. Suzuki. Learning Bayesian belief networks based on the minimum description length principle: An efficient algorithm using the b & b technique. In *Proceedings of the 13th International Conference on Machine Learning (ICML 1996)*, ICML'96, pages 462–470. Morgan Kaufmann Publishers Inc., 1996.

[87] J. Tian. A branch-and-bound algorithm for MDL learning Bayesian networks. In C. Boutilier and M. Goldszmidt, editors, *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence (UAI 2000)*, pages 580–588. Morgan Kaufmann, 2000.

[88] J. Tian. Identifying direct causal effects in linear models. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005)*, pages 346–352. AAAI Press, 2005.

[89] S. Triantafillou and I. Tsamardinos. Score-based vs constraint-based causal learning in the presence of confounders. In *Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence (UAI 2016)*, volume 1792 of *CEUR Workshop Proceedings*, pages 59–67. CEUR-WS.org, 2016.

[90] S. Triantafilou, I. Tsamardinos, and I. G. Tollis. Learning causal structure from overlapping variable sets. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9 of *JMLR Proceedings*, pages 860–867. JMLR.org, 2010.

[91] I. Tsamardinos, L. E. Brown, and C. F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

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

[93] P. van Beek and H. Hoffmann. Machine learning of Bayesian networks using constraint programming. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP 2015)*, volume 9255 of *Lecture Notes in Computer Science*, pages 429–445. Springer, 2015.

[94] T. S. Verma and J. Pearl. An algorithm for deciding if a set of observed independencies has a causal explanation. *Computing Research Repository*, abs/1303.5435, 2013.

[95] A. Wiesel, Y. C. Eldar, and A. O. H. III. Covariance estimation in decomposable Gaussian graphical models. *IEEE Transactions on Signal Processing*, 58(3):1482–1492, 2010.

[96] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, 2011.

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

[98] Zhalama, J. Zhang, F. Eberhardt, W. Mayer, and M. J. Li. ASP-based discovery of semi-Markovian causal models under weaker assumptions. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 1488–1494. ijcai.org, 2019.

[99] J. Zhang. Causal reasoning with ancestral graphs. *Journal of Machine Learning Research*, 9:1437–1474, 2008.

[100] J. Zhang. On the completeness of orientation rules for causal discovery in the presence of latent confounders and selection bias. *Artificial Intelligence*, 172(16-17):1873–1896, 2008.