ECAI 2020
G.D. Giacomo et al. (Eds.)
© 2020 The authors and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms
of the Creative Commons Attribution Non-Commercial License 4.0 (CC BY-NC 4.0).
doi:10.3233/FAIA200400

Learning Chordal Markov Networks via Stochastic Local Search

Kari Rantanen and **Antti Hyttinen** and **Matti Järvisalo** [1]

Abstract. We present a novel stochastic local search (SLS) approach for the computationally hard task of finding a chordal Markov network structure that maximizes a given scoring function (CMSL for short). Current state of the art in exact algorithms for CMSL only scale up to circa 20 variables. Beyond this, the scalability of exact approaches is obstructed by the drastically increasing number of clique scores which grows exponentially with the number of variables. We show that, in practice, using only a fraction of the running times of the exact approaches, our SLS approach provides optimal or very close to optimal solutions for instance sizes that are within the reach of exact algorithms. Furthermore, we propose an on-the-fly clique score computation approach that enables scaling up our SLS approach towards hundreds of variables. In particular, on-the-fly score computation circumvents the need to enforce low treewidth bounds, which enable pre-computation of scores before search, but which also may severely limit the accuracy of the learned models.

1 INTRODUCTION

Graphical models are central tools for dealing with uncertainty in AI, and allow for leveraging on independencies for computational and statistical advantage in various data analysis tasks [1, 40, 20]. In this context, learning the structure of a graphical model representing data, i.e., structure learning, is an important computational problem. In this paper we focus on the structure learning task for *chordal Markov networks* (or chordal/triangulated Markov random fields or decomposable graphs), a central class of undirected graphical models [10, 41, 22, 20]. This problem, chordal Markov network structure learning (CMSL), is computationally challenging; e.g., finding a maximum likelihood chordal Markov network with bounded structure complexity (clique size) is known to be NP-hard [35]. In particular, we consider the score-based setting, giving rise to a challenging combinatorial optimization problem, and focus on the challenge of scaling up search for good quality chordal Markov network structures fast to hundreds of variables.

Algorithms developed for CMSL can be categorized as in-exact (or incomplete) approaches [24, 12, 29, 2, 26, 5, 34, 18, 13] that are not guaranteed to find optimal solutions, and exact approaches [7, 14, 15, 3, 30] that, given enough resources, provide provably optimal solutions. While early work on algorithms for CMSL focused on in-exact approaches, more recently the development of practical *exact* algorithms for CMSL has received noticeably attention. Exact approach are capable of finding chordal Markov network structures that optimally fit the data in the score-based setting [9, 3, 14, 15, 30]. However, emphasizing the challenging nature of CMSL, the current state of the art in exact approaches does not generally scale up to more than 20 variables. Furthermore, the sheer number of clique scores—exponential in the number of variables—poses challenges even for scaling up in-exact approaches.

In this paper, we provide new solutions to the challenge of scaling up search for accurate chordal Markov network structures fast to hundreds of variables. Standardly used structural restrictions—both in the context of in-exact and exact approaches—for enabling search over large number of nodes come in terms of enforcing treewidth bounds [31, 18, 2, 5, 27, 4, 34, 33]. While a main motivation in restricting the treewidth of networks to be learned by limiting the size of cliques considered is in guarantees on the running time of exact inference over the learned networks structures [8, 19], in practice for large networks a significantly strong bound on the size of cliques needs to be enforced in order to enable the computation of cliques scores, which may severely limit the accuracy of the learned models. Here we propose an alternative solution to this issue—avoiding enforcing instance-inspecific structural restrictions—in the context of stochastic local search.

In particular, we develop a stochastic local search (SLS) approach to CMSL, making use of large neighbourhoods defined in terms of various types of structural changes to chordal Markov network structures. Furthermore, we propose on-the-fly score computation as a means of breaking through the scalability barrier imposed by the pre-computation of scores before the actual search for chordal Markov network structures. On-the-fly score computation is especially suited for SLS-style search, and allows for making early progress towards accurate solutions from the start.

As evidence of the practical value of our approach, we show empirically that using only a fraction of the running times of the exact approaches, our SLS approach surprisingly provides optimal or very close to optimal solutions for instances that are within the reach of exact algorithms. Furthermore, the SLS approach together with on-the-fly score computation allows for scaling search up to hundreds of variables. In particular, with on-the-fly score computation only scores needed during local search steps are computed, thereby allowing search to proceed guided by search heuristics without instance-inspecific structural restrictions typical to in-exact approaches.

The rest of this paper is organized as follows. After background on CMSL and earlier algorithmic approaches to the problem (Section 2), we turn to our main contributions, describing a new SLS approach to CMSL (Section 3) and proposing, in conjunction with the SLS approach, on-the-fly score computation as a means to scaling up to hundreds of variables without need for pre-computing of the exponentially many scores or imposing structural constraints on the search space (Section 4). Results from an extensive empirical evaluation of the approach are provided in Section 5.

[1] HIIT, Department of Computer Science, University of Helsinki, Finland. Emails: firstname.lastname@helsinki.fi

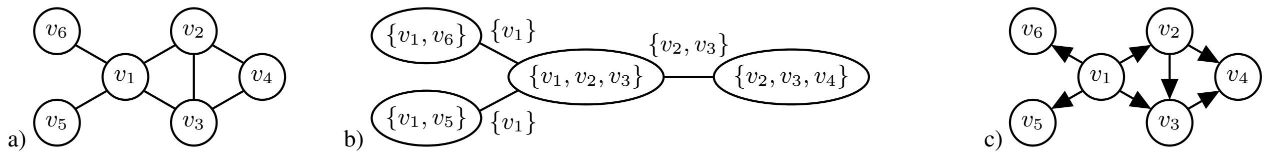

Figure 1. Different views on chordal Markov network structures: a) chordal undirected graph, b) clique tree, c) decomposable directed acyclic graph.

2 CHORDAL MARKOV NETWORK STRUCTURE LEARNING

In this section, we will provide necessary background on the chordal Markov network structure learning problem and shortly review earlier developed algorithmic approaches for the problem.

2.1 CMSL

A Markov network structure is represented by an undirected graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ is the set of vertices and E the set of undirected edges. This structure represents independencies $v_i \perp\!\!\!\perp v_j | S$ according to the undirected separation property: v_i and v_j are separated given set S if and only if all paths between them go through a vertex in set S. The undirected graph is *chordal* iff every (undirected) cycle of length greater than three contains a *chord*, i.e., an edge between two non-consecutive vertices in the cycle. Figure 1 a) shows an example.

Here we focus on the task of finding a chordal graph G that maximizes posterior probability $P(G|D) = P(D|G)P(G)/P(D)$, where D denotes the i.i.d. data set. As we assume a uniform prior over chordal graphs, this boils down to maximizing the marginal likelihood $P(D|G)$. Dawid et al. have shown that the marginal likelihood $P(D|G)$ for chordal Markov networks can be calculated using a clique tree representation [10, 11]. A clique C is a fully connected subset of vertices. A clique tree for an undirected graph $G = (V, E)$ is an undirected tree such that

I. $\bigcup_i C_i = V$,
II. if $(v_\ell, v_k) \in E$, then there is a C_j s.t. $\{v_\ell, v_k\} \subseteq C_j$, and
III. the running intersection property holds: whenever $v_k \in C_i$ and $v_k \in C_j$, then v_k is also in every clique on the unique path between C_i and C_j.

The separators are the intersections of adjacent cliques in a clique tree. Figure 1 b) shows an example. The marginal likelihood factorizes according to the clique tree: $P(D|G) = \prod_i P(C_i)/\prod_j P(S_j)$ (assuming positivity and that the prior factorizes) [9]. The marginal likelihood $P(S)$ for a set S of random variables can be calculated with suitable priors; in this paper we consider discrete data using a Dirichlet prior. If we denote $s(S) = \log P(S)$, CMSL can be cast as the following optimization problem:

$$\max_{G \in \mathcal{G}} \sum_{C_i} s(C_i) - \sum_{S_j} s(S_j),$$

where $\mathcal{G}$ denotes the class of chordal Markov networks, C_i and S_j are the cliques and the separators of the graph G.

Example 2.1. The marginal log-likelihood of the graph in Figure 1 a) can be calculated using the clique tree presentation in Figure 1 b) as $s(\{v_1, v_6\}) + s(\{v_1, v_5\}) + s(\{v_1, v_2, v_3\}) + s(\{, v_2, v_3, v_4\}) - s(\{v_1\}) - s(\{v_1\}) - s(\{v_2, v_3\})$.

By considering cliques of size at most k when solving CMSL results in the structural restriction that the learned network structures have treewidth at most $k - 1$.

2.2 Algorithmic Approaches to CSML

Before describing our SLS approach to CSML, we overview earlier developments in algorithm approaches to the problem. Beyond trees studied by Chow and Liu [7], both in-exact and exact learning of chordal Markov networks structure have been considered.

The literature on in-exact approaches has focused in particular on networks with a hard bound on the treewidth in order to facilitate fast exact inference. Early approaches considered e.g. greedy procedures [24, 12]. Della Pietra et al. and Bach et al. developed iterative scaling approaches [29, 2]. Narasimhan et al. and Chechetka et al. considered PAC-learning in this context [26, 5], Shahaf et al. used Bethe approximation and graph cuts [34]. Kumar et al. provided a convex relaxation [18]. Several works have also examined minor variations of the graphical model class [17, 28, 37]. Beyond hard treewidth bounds, Gogate et al. proposed learning high-treewidth chordal Markov networks based on mutual information and showed that by using context-specific independence and determinism tractable exact inference is possible even with higher treewidths [13].

For solving CMSL exactly, Janhunen et al. proposed a constraint satisfaction approach [14]. The exact dynamic programming approach of Kangas et al. (Junctor) scales up considerably better, to 17 variables [15, 16]. The state-of-the-art exact Bayesian network learning system (GOBNILP) based on integer linear programming can learn also chordal Markov networks if the treewidth is restricted [3, 36]. The branch and bound approach of Rantanen et al. (BBMarkov) can find provably optimal solutions for some 20 variable instances [30].

We will next describe a simple yet effective stochastic local search approach to CSML. As we will show through extensive comparisons against the exact methods, the approach can learn optimal or close to optimal networks without hard treewidth bounds that might end up jeopardizing the quality of the learned models.

3 STOCHASTIC LOCAL SEARCH FOR CMSL

In this section we describe our stochastic local search for learning chordal Markov networks.

3.1 Search Over Undirected Graphs

Our search uses undirected graphs as solution candidates.

We begin by explaining the standard method that we use to compute the score of an undirected graph $G = (V, E)$ and to verify its chordality. First, we determine the lexicographic ordering of G using a breadth-first search [32]. The algorithm works by keeping a track of a collection Σ that partitions the vertices V. Starting with $\Sigma = \{V\}$, the algorithm repeatably selects the lexicographically least vertex $v \in V$ from the first partition of Σ, removes it from the partition, and then splits each set $S \in \Sigma$ into two new partitions; one containing the neighbours of v in S and the other containing non-neighbours of v in S. The partition with the neighbours is placed prior to the other one in Σ, and any empty sets resulting from this process are removed. Once Σ is empty, then the order in which we chose vertices in the algorithm is the lexicographic ordering of G.

Example 3.1. Consider the graph shown in Figure 1 a). We can apply the lexicographic breadth-first search [32] in the following way.
(1.) Start with $\Sigma = \{\{v_1, v_2, v_3, v_4, v_5, v_6\}\}$ and we pick node v_1.
(2.) Now $\Sigma = \{\{v_2, v_3, v_5, v_6\}, \{v_4\}\}$ and we pick node v_2.
(3.) Now $\Sigma = \{\{v_3\}, \{v_5, v_6\}, \{v_4\}\}$ and we pick node v_3.
(4.) Now $\Sigma = \{\{v_5, v_6\}, \{v_4\}\}$ and we pick the node v_5.
(5.) Now $\Sigma = \{\{v_6\}, \{v_4\}\}$ and we pick v_6, and finally v_4.
The final lexicographic ordering is $v_1, v_2, v_3, v_5, v_6, v_4$.

The lexicographic ordering corresponds to orienting the edges of G. Let σ_i be the position of $v_i \in V$ in the lexicographic ordering. For each vertex $v_i \in V$, have its parent set $pa(v_i)$ contain exactly the neighbours of v_i in G that occur prior to it in the ordering. That is, $pa(v_i) = \{v_j : v_j \in V, \{v_i, v_j\} \in E, \sigma_j < \sigma_i\}$.

Example 3.2. Consider Figure 1a) again. We can orient the edges using the lexicographic ordering obtained in the previous example $(v_1, v_2, v_3, v_5, v_6, v_4)$. We get the (decomposable) DAG shown in Figure 1 c) where edges are oriented according to the ordering.

The undirected graph is chordal if and only if for all $v \in V$ we either have $pa(v) = \emptyset$ or $pa(v) \setminus \{w\} \subseteq pa(w)$ where w is the closest parent of v in the lexicographic ordering; i.e., $w = \arg\max_{v_i \in pa(v)} \sigma_i$. The score of G is obtained simply by summing up scores for each vertex in the corresponding (decomposable) DAG:

$$\sum_{i=1}^{n} s(pa(v_i) \cup \{v_i\}) - s(pa(v_i)).$$

3.2 Local Search for CMSL

The overall structure of the search is outlined in Algorithm 1. At every search step, we keep track of two feasible solutions G and G^*. G^* is the incumbent highest-scoring solution that has been found so far during the search. The algorithm traverses through the search space by moving from a current solution G to another solution in the neighbourhood of G (as detailed in Section 3.3). Furthermore, the search is subjected to restarts using a tailored restart schedule (as detailed in Section 3.4).

The search starts by initializing G and G^* to an initial solution on line 2. On line 3, we enter the optimization loop which lasts until some given criteria for stopping is satisfied (if any). On line 4, we update the incumbent best solution G^* if the score of G exceeds that of G^*. Then, on line 6 we check whether the search has progressed recently (as detailed in Section 3.4). If it has (line 9), we continue the search by setting G to be one of its best stochastic neighbours chosen at random (Section 3.3). Otherwise (line 7), we restart the search by setting G into some randomly generated solution.

3.3 Local Search Neighbourhoods

A neighbourhood $N = \{G_1, \ldots, G_m\}$ is a set of undirected graphs that are obtained by performing *neighbourhood operations* on a current solution (chordal undirected graph) G. Our SLS method employs substantially large neighbourhoods based on three types of structural changes targeted at cliques, nodes, and edges. (We will later show empirically that this combination gives substantial improvements compared to using subsets of the three types of operations.)

We will define the neighbourhood operations using $\mathcal{C}$, the set of maximal cliques of G. This is because the maximal cliques uniquely identify a chordal graph, and any set of cliques (not necessarily maximal) can be unambigously interpreted as an undirected graph.

Clique-specific neighbourhood $N_{\mathcal{C}}(G, C)$ where $C \in \mathcal{C}$: For all $v \in V$, let $G_v \in N(G, C)$. If $v \in C$, then G_v is formed from G by deleting all edges from v to nodes in $C \setminus \{v\}$; otherwise, G_v is formed from G by adding edges from v to every node in C.

Example 3.3. Consider the graph in Figure 1 a) as G. One of its maximal cliques is $C = \{v_2, v_3, v_4\}$. The clique-specific neighbourhood for C contains all the undirected chordal graphs based on G where one vertex is either added to C or removed from C. Hence C can take the following modified forms: $\{v_2, v_3\}, \{v_3, v_4\}, \{v_2, v_4\}, \{v_2, v_3, v_4, v_1\}$. The neighbours where C is $\{v_2, v_3, v_4, v_5\}$ or $\{v_2, v_3, v_4, v_6\}$ are excluded since the corresponding graphs would not be chordal.

Node-specific neighbourhood $N_V(G, v)$ where $v \in V$: For all $C \in \mathcal{C}$, let $G_C \in N(G, v)$. If $v \in C$, then G_C is formed from G by deleting all edges from v to nodes in $C \setminus \{v\}$; otherwise, G_C is formed from G by adding edges from v to every node in C.

Example 3.4. Consider the graph in Figure 1 a) as G. The node-specific neighbourhood for v_4 contains all the undirected chordal graphs based on G where v_4 is either added to a maximal clique in G or removed from a maximal clique in G. This gives us two neighbours. First, v_4 is added to $\{v_1, v_2, v_3\}$. The neighbour contains a clique $\{v_1, v_2, v_3, v_4\}$ instead of $\{v_1, v_2, v_3\}$ and $\{v_2, v_3, v_4\}$. Second, v_4 is removed from $\{v_2, v_3, v_4\}$. The neighbour contains the cliques $\{v_4\}$ and $\{v_2, v_3\}$ instead of $\{v_2, v_3, v_4\}$.The neighbours where v_4 is added to $\{v_1, v_5\}$ or $\{v_1, v_6\}$ are excluded since the corresponding graphs would not be chordal.

Edge-specific neighbourhood $N_E(G, v)$ where $v \in V$: For all $w \in V$ with $w \neq v$, let $G_w \in N_E(G, v)$. If $(v, w) \in E$, then G_w is formed from G by deleting the edge (v, w); otherwise, G_w is formed from G by adding the edge (v, w).

Example 3.5. Consider the graph in Figure 1 a) as G. The edge-specific neighbourhood relative to node v_3 contains all the undirected chordal graphs based on G where an edge is added between v_3 and another vertex or an edge is removed between v_3 and another vertex. The possible edges to be added are (v_3, v_5) and (v_3, v_6). The possible edges to be removed are (v_3, v_1) and (v_3, v_4). The neighbour with the edge (v_2, v_3) removed is excluded since it is not chordal.

Algorithm 1 The core structure of the local search

```
1: function SEARCH(variables $V$, score function $s$)
2:     $G, G^* \leftarrow$ INITIALSOLUTION
3:     while a stopping criteria is not met do
4:         if $s(G^*) < s(G)$ then
5:             Update $G^* \leftarrow G$
6:         if no recent progress then
7:             $G \leftarrow$ RANDOMFEASIBLESOLUTION()
8:         else
9:             $G \leftarrow$ BESTSTOCHASTICNEIGHBOUR($G$)
```

Given a solution G, we construct a stochastic neighbourhood in the following way. Let $v \in V$ be a randomly chosen node and let C be a randomly chosen maximal clique in G. Then our stochastic neighbourhood is

$$N_G = N_C(G, v) \cup N_V(G, C) \cup N_E(G, v).$$

Moreover, let F_G be a subset of N_G containing only feasible (chordal) solutions. Then we can define $\textsc{BestStochasticNeighbour}(G)$ on line 9 of Algorithm 1 to return a random element from the set

$$\arg\max_{G' \in F_G} s(G').$$

3.4 Restart Schedules and Initial Solutions

A well-known general issue with stochastic local search is that the search can get stuck into specific areas of the search space around local optima without finding further better solutions. Here we propose an adaptive restart schedule in the context of our CMSL SLS approach. We keep track of an additional solution $\hat{G}$ which stores the best solution found after the last (re)start. Similarly to G^*, we update $\hat{G} \leftarrow G$ whenever $s(\hat{G}) < s(G)$. Then, if the search spends more than a predetermined number of iterations T without improving $\hat{G}$, we restart the search by initializing G into some random feasible solution and resetting $\hat{G} \leftarrow G$. In other words, we postpone restarting as long as the search makes progress towards better solutions. A larger number of iterations between restarts can result in better solutions per restart, but on the other hand a good randomly generated G can potentially lead to even better solutions. With this intuition, our goal is to achieve a good balance between the number of restarts (exploring the global search space) and the number of iterations between restarts (exploring the local search spaces from the generated starting positions).

To obtain the first initial solution in the beginning of Algorithm 1 on Line 2, G and G^* can be initialized to an optimal chordal Markov network structure of treewidth at most one. This is achieved in low polynomial time using the classical Chow-Liu algorithm [7].

We generate a random new solution to start from at each restart as follows. First, we construct a list containing all the nodes V and shuffle it into a random order. Then, the maximal cliques of the new solution are formed by partitioning the shuffled list into consecutive disjoint sets with random sizes from one to three. Since the cliques are disjoint, the chordality of the graph is guaranteed. We limit the maximum size of the randomly generated cliques here to avoid score computation for overly large cliques, see Section 4.

4 ON-THE-FLY SCORE COMPUTATION

Pre-computation of all clique scores—even of limited size, resulting in learning bounded-treewidth chordal Markov networks—is a noticeable obstacle for scaling up CMSL algorithm to larger networks (with hundreds of variables). Regardless of the efficiency of the search algorithm, pre-computation of clique scores dominates runtimes, and even worse, with hundreds of variables simply storing all clique scores surpasses reasonable memory requirements. As a solution to this fundamental issue, we propose an approach to computing clique scores on-the-fly during search, building on score computation strategies in other contexts [38, 6]. This is particularly suitable in the context of stochastic local search, where search progresses with local moves towards better solutions, hence allowing us to completely avoid computing many of the less important clique scores.

Algorithm 2 A recursive procedure for computing a score for a clique

```
1: function ComputeScore(data $D$, clique $C = \{v_1, \ldots, v_n\}$,
   depth $i$, equivalent sample size $\alpha$)
2:     if $i > |C|$ then return 0
3:     $s \leftarrow \log\Gamma(\frac{\alpha}{q_i}) - \log\Gamma(|D| + \frac{\alpha}{q_i})$
4:     for $k \in \{1, \ldots, r_i\}$ do
5:         $D_k \leftarrow$ all the samples of $D$ where $v_i$ takes its $k$th value
6:         $s \leftarrow s + \log\Gamma(|D_k| + \frac{\alpha}{r_i \cdot q_i}) - \log\Gamma(\frac{\alpha}{r_i \cdot q_i})$
7:         if $D_k \neq \emptyset$ then
8:             $s \leftarrow s + \textsc{ComputeScore}(D_k, C, i+1, \alpha)$
    return $s$
```

Let $C = \{v_1, \ldots, v_n\}$ be a clique and let r_i be the arity (number of unique values) of each $v_i \in C$ in data D. The BDeu score for C is obtained with

$$s(C) = \sum_{i=1}^{n}\sum_{j=1}^{q_i} \log\Gamma\left(\frac{\alpha}{q_i}\right) - \log\Gamma\left(\sum_{k=1}^{r_i} N_{i,j,k} + \frac{\alpha}{q_i}\right) + \sum_{k=1}^{r_i} \log\Gamma\left(N_{i,j,k} + \frac{\alpha}{r_i \cdot q_i}\right) - \log\Gamma\left(\frac{\alpha}{r_i \cdot q_i}\right), \quad (1)$$

where $q_i = r_1 \cdot r_2 \cdots r_{i-1}$, integer $N_{i,j,k}$ is the number of samples in D in which v_i takes its kth value and the variables $v_1, \ldots, v_{i-1}$ take their jth configuration and α is the prior parameter (equivalent sample size) for BDeu scoring.

The score computation is detailed in Algorithm 2 as an adaptation of the DMScore approach of Kangas et al. [16]. At its core this is a slightly simplified version of the AD-tree algorithm commonly used for computing local scores for structure learning tasks [25]. Instead of constructing an explicit tree structure to store information in, an implicit tree is constructed recursively and certain optimizations such as the subtraction trick are omitted.

In Algorithm 2, the data D consists of the set of samples from the given CSV file and hence $|D|$ is the sample count. The recursion depth i determines which variable v_i to process next. For computing the BDeu score for clique C with a chosen equivalent sample size α, one would call the procedure with $\textsc{ComputeScore}(D, C, 1, \alpha)$. The algorithm works by recursively partitioning the data D according to what values the samples give to the variables v_i on each step. Then the sizes of the partitioned datasets automatically yield the required $N_{i,j,k}$ counts for each step. Importantly, for limited data set sizes, the sample size of D_k becomes very low or even zero (Line 7) after only a few recursion calls, making Algorithm 2 efficient in this context.

In order to avoid recomputing same clique scores repeatedly, we cache the scores of already-computed cliques in memory. Thus whenever $\textsc{ComputeScore}$ is called with same arguments as earlier in the search, we simply fetch the corresponding score from the cache.

For very large cliques the q_i values in Equation 1 can get very large, to the extent that computing the score for a clique becomes difficult using standard fixed-size datatypes. We circumvent this problem in the following way: If a graph has a clique for which some q_i would become too large, then we declare the score of that graph to be $-\infty$.

5 EMPIRICAL EVALUATION

We proceed with an in-depth empirical evaluation of the effectiveness of our SLS approach and on-the-fly score computation on CMSL instances obtained from real-world datasets, focusing on three points of views.

(i) We compare our SLS approach to recently proposed state-of-the-art exact algorithms for CMSL. The results show that our SLS approach consistently finds optimal or close to optimal solutions

Table 1. Comparsion of lsmarkov to the exact approaches with $n = 17$. Highest scores and among those the shortest times are in bold.

		lsmarkov		**Junctor**		**BBMarkov**	
Name	n	**Score**	**Time (s)**	**Score**	**Time (s)**	**Score**	**Time (s)**
autos	17	**-1127.9**	**264**	**-1127.9**	2508	-1191.3	3313
water10000	17	**-97079.6**	**13**	**-97079.6**	2183	-97649.0	48
insurance10000	17	**-100757.3**	**501**	**-100757.3**	2691	-102030.4	71
steel	17	**-12960.2**	**310**	**-12960.2**	1875	-13226.3	2950
horse	17	**-3013.4**	**0**	**-3013.4**	2091	-3046.6	139
flag	17	**-1893.3**	**1**	**-1893.3**	2474	-1975.7	401
Epigenetics	17	**-100284.8**	**514**	**-100284.8**	1925	-102275.6	2255
wdbc	17	**-3954.5**	**14**	**-3954.5**	2031	-4079.4	1513
mildew10000	17	**-190359.0**	**0**	**-190359.0**	2837	-190371.5	1603
soybean	17	**-2252.0**	**6**	**-2252.0**	1939	-2397.0	165
alarm10000	17	**-38843.1**	**1**	**-38843.1**	2505	-38869.2	39
bands	17	**-1797.0**	**96**	**-1797.0**	2534	-1928.8	13
connect-4	17	**-488636.3**	**450**	**-488636.3**	2742	-496106.1	88
spectf	17	**-3022.6**	**1**	**-3022.6**	2383	-3039.3	2279
sponge	17	**-569.2**	**612**	**-569.2**	1920	-578.2	1621
barley5000	17	**-84468.7**	**433**	**-84468.7**	2016	-93463.9	1876
hailfinder10000	17	**-162384.4**	**0**	**-162384.4**	2623	-164114.2	132
lung-cancer	17	**-481.2**	**16**	**-481.2**	1870	-490.3	2310
promoters	17	**-2362.0**	**0**	**-2362.0**	1862	-2407.1	2826
carpo10000	17	**-35271.6**	**3**	**-35271.6**	1983	-35281.5	21
kddcup	17	-15854.9	87	**-15854.6**	2266	-16000.3	2709
dota2	17	**-121215.4**	**0**	**-121215.4**	2468	**-121215.4**	68
msweb	17	**-14703.9**	**82**	**-14703.9**	1797	-14778.5	3568

in a fraction of the runtimes of exact approaches on instances in reach of the exact approaches (similarly as [21] for BNs).

(ii) We compare the proposed on-the-fly score computation to the more standard approach of pre-computing a restricted subset of clique scores. The results show that on-the-fly score computation results in better solutions faster without needing to restrict the considered cliques before search, scaling up to hundreds of variables.

(iii) We evaluate the marginal contributions of the three types of neighbourhoods employed in our SLS approach on the effectiveness of the approach.

We implemented the SLS approach in C++, and refer to this prototype as lsmarkov. The implementation is available in open source online. Regarding the restart schedule (Section 3.4), our implementation uses $T = 1000$. That is, the search is restarted whenever thousand iterations have passed by without improvements to the current incumbent solution $\hat{G}$. We computed the input clique scores using the DMScore program [16], apart from GOBNILP and lsmarkov. The score computation times are included in all presented results. For the evaluation, we used a total of 23 real-world datasets used as standard benchmarks for exact approaches [23, 39, 3]. For computing clique scores, we used the BDeu score with equivalent sample size 10. The experiments were run under Debian GNU/Linux on 2.83-GHz Intel Xeon E5440 nodes with 32-GB RAM.

5.1 Comparison with Exact Approaches

In terms of state-of-the-art exact approaches, we will compare the performance of lsmarkov with those of GOBNILP [3] (version 1.6.3, using SoPlex 2.2.0 as the internal IP solver) implementing an integer programming branch-and-cut approach to CMSL; Junctor [15], implementing a state-of-the-art DP approach to CMSL; and BBMarkov [30], a recent branch-and-bound approach to CMSL. (While in-exact approaches to CMSL have been previously proposed [18, 34, 5, 17], we note that despite our efforts we were unable to obtain implementations of these approaches for comparison.) Differentiating the exact approaches, GOBNILP and BBMarkov can report improving solutions during search already before outputting a provably optimal solution, unlike Junctor. We will report on the costs of best solutions found by the approaches with a per-instance time limit of 1 hour.

We first compare the quality of solutions (in terms of the scores of networks found) obtained with lsmarkov with the scores of those obtained with the exact approaches Junctor and BBMarkov without bounding treewidth of the learned networks. (We do not report results for GOBNILP for this comparison as it could not provide any solutions.) For this comparison, we restricted the number of variables considered in the datasets to $n = 17$ based on the fact that the running times for finding provably optimal solutions for the best-performing exact approaches tend to be around 30-60 minutes on instances with $n = 17$. The results are shown in Table 1. We observe that apart from one exception lsmarkov in fact produced solutions with the same cost (up to numerical precision) as Junctor, i.e., lsmarkov was able to find an optimal solution for almost every instance. At the same time, lsmarkov uses consistently less time (using only up to 10 minutes) for finding an optimal solution than Junctor. BBMarkov in most cases is unable to find an optimal solution within the 1-hour time limit.

A way of scaling up GOBNILP and BBMarkov to some extent to instances with higher number of nodes n is to considerably limit the size of cliques for which scores are considered, thereby restricting the search space to networks with low treewidth. Taking into account all variables in each of the datasets, Table 2 provides a comparison of lsmarkov (without limiting the size of cliques considered), GOBNILP with clique-size bound 3, and BBMarkov with clique-size bounds 3 and 5. GOBNILP was unable to provide any solutions on clique-size bound 5, and Junctor ran out of memory on all instances for both of the clique-size bounds. We observe that GOBNILP is unable to prove any solutions for instances with $n > 60$ even with cliques-size bound

Table 2. Comparison of lsmarkov to the exact approaches without limiting n. Here lsmarkov was ran without treewidth bound while the other algorithms were tested enforcing clique size (cs) $\leq$3 and $\leq$5. Highest scores and among those the shortest time are in bold.

		lsmarkov		**GOBNILP** $cs \leq 3$		**BBMarkov** $cs \leq 3$		**BBMarkov** $cs \leq 5$	
Name	n	**Score**	**Time (s)**	**Score**	**Time (s)**	**Score**	**Time (s)**	**Score**	**Time (s)**
autos	26	**-1513.2**	**100**	-1641.8	73	-1866.7	17	-1921.7	21
water10000	26	-128910.9	769	**-128909.2**	**17**	-130327.0	18	-130560.5	31
insurance10000	27	**-134175.5**	**290**	-134701.2	92	-159329.4	34	-159329.4	55
steel	28	**-18604.4**	**425**	-19908.1	561	No	-	No	-
horse	28	**-4503.0**	**513**	-4511.9	19	No	-	No	-
flag	29	**-2720.0**	**339**	-2747.8	47	No	-	No	-
Epigenetics	30	**-177782.8**	**854**	-188915.2	230	No	-	No	-
wdbc	31	**-6654.7**	**602**	-6771.5	1762	No	-	No	-
mildew10000	35	-454360.8	545	**-454312.8**	**133**	No	-	No	-
soybean	36	**-2932.2**	**715**	-3332.8	1354	No	-	No	-
alarm10000	37	**-105155.3**	**408**	-105424.7	378	No	-	No	-
bands	39	**-5099.6**	**350**	-5145.1	293	No	-	No	-
connect-4	43	**-1008451.1**	**171**	-1043546.0	573	No	-	No	-
spectf	45	-8055.0	355	**-8047.4**	**3067**	No	-	No	-
sponge	45	**-1788.1**	**89**	-1813.1	4264	No	-	No	-
barley5000	48	**-266863.3**	**537**	-269916.7	1032	No	-	No	-
hailfinder10000	56	**-497216.1**	**758**	-497784.1	9134	No	-	No	-
lung-cancer	57	**-1248.9**	**262**	-1366.2	8024	No	-	No	-
promoters	58	**-8326.5**	**3**	**-8326.5**	1693	No	-	No	-
carpo10000	60	**-173880.7**	**765**	-176671.8	9344	No	-	No	-
kddcup	60	**-72971.4**	**207**	No	-	No	-	No	-
dota2	115	**-442078.8**	**20**	No	-	No	-	No	-
msweb	235	**-51596.0**	**881**	No	-	No	-	No	-

3, and BBMarkov is unable to provide any solutions for $n \geq 28$. On the other hand, lsmarkov, using on-the-fly score computation and a per-instance time limit of 15 minutes, can find on a majority of the instances better solutions than GOBNILP (as at least as good solutions on all instances).

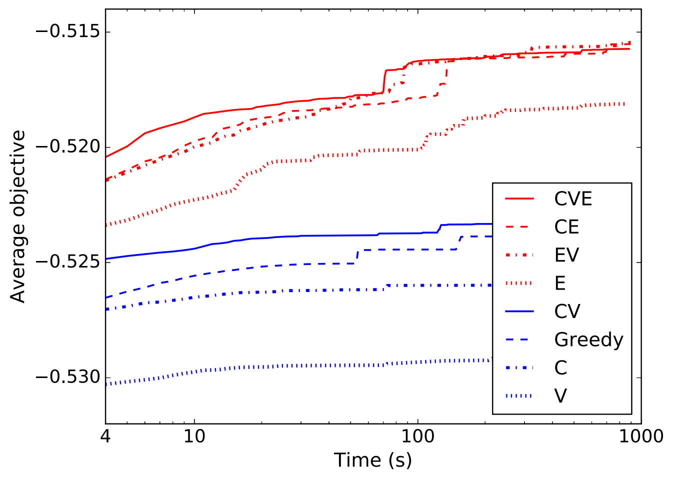

Figure 2. Impact of neighbourhoods on lsmarkov w.r.t. the average objective, i.e., the sum of the best scores during lsmarkov search at different time points divided by the number of instances (23), and the numbers of variables and samples in each instance. The plot starts at 4 seconds as the earliest point when a score has been obtained for every instance.

5.2 Impact of Neighbourhoods

Next, we investigate the importance of the three different types of neighbourhoods (recall Section 3.3) on the performance of lsmarkov. As a baseline, we also consider a simplistic greedy version, using a brute-force greedy edge-specific neighbourhood, selecting at each step of the search a random element from

$$\arg\max_{G' \in \bigcup_{v \in V} N_E(G,v)} s(G').$$

That is, the greedy neighbourhood consists of the highest-scoring solutions that can be obtained by performing a single edge addition or removal from a given graph G. When using a greedy neighbourhood instead of a stochastic one, the score of G can only get worse after the search has reached a local optimum. Improving on the naive greedy approach, we include also restarts in this baseline: a restart is performed every time the score of G is worse than at the previous iteration. Moreover, also in the case of this greedy baseline search, we still ensure at all times that the resulting graphs are chordal.

Figure 2 shows a comparison of all possible combinations of the edge-specific (E), node-specific (V) and clique-specific (C) neighbourhoods in terms of the average score obtained from solving each of the 23 instances up to a given point of time using each of the combined neighbourhoods, with the greedy neighbourhood strategy as a baseline. We observe that the combination CVE of all of the three neighbourhoods clearly outperforms the greedy neighbourhood. The edge-specific neighbourhood has the greatest marginal contribution. However, both C and V provide additional orthogonal performance gains, as well. Furthermore, Figure 3 shows that the performance variation due to stochasticity does not overweight the observed performance differences due to using different neighbourhoods: the variation in the performance of CVE under seven different seeds to the random number generator is low with respect to the reference E.

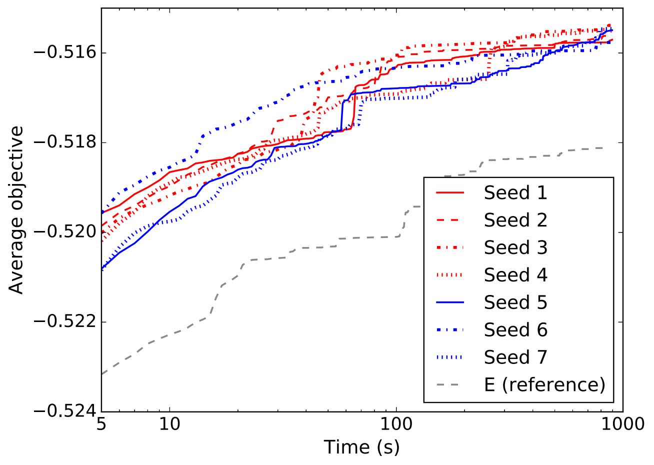

Figure 3. Variation in the performance of lsmarkov under different seeds to the random number generator when employing the CVE neighbourhood strategy, in comparison to the reference E neighbourhood strategy.

5.3 Impact of On-the-Fly Score Computation

Finally, we investigate the impact of score computation strategy. For this, we compare lsmarkov with on-the-fly score computation to lsmarkov with pre-computation of scores under low treewidth bounds.

The results are shown in Table 3. We observe that on a majority of the instances, employing on-the-fly score computation results in finding better models than by enforcing the strict, instance-inspecific structural bounds. Furthermore, on-the-fly score computation allows for scaling up to hundreds of variables while allowing the search itself guide the score computations as needed towards better solutions. Even a slight increase in the treewidth bound results in score precomputation to time out. As shown in Figure 4, we also observe that the number of scores needed to compute on-the-fly during search is indeed quite small even compared to the number of precomputed scores under low treewidth bounds.

The time spent in on-the-fly score computation depends, in addition to the number of variables, on the sample size used. On the benchmarks in Table 3 time spent in on-the-fly score computation was around 6% of the 15-min per-instance time limit (and around 11% of the time it took for lsmarkov to find the best solution).

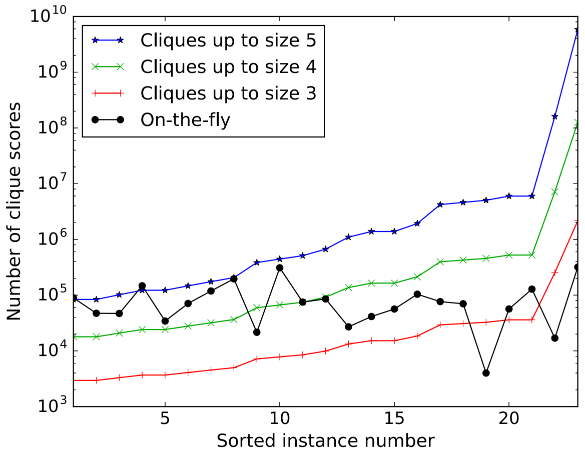

Figure 4. Number of clique scores computed by lsmarkov with on-the-fly score computation during search for the instances in Table 2 sorted wrt number of variables, with the total number of clique scores up to a clique size (3, 4, 5) for comparison.

6 CONCLUSIONS

We presented a simple yet well-performing stochastic local search approach to learning chordal Markov networks structures that scales towards hundreds of variables. Applying large neighbourhoods based on multiple types of structural changes, an adaptive restart schedule, and on-the-fly score computation, we showed that the approach provides essentially optimal solutions for network sizes that are within the reach of current state-of-the-art exact algorithms for the problem. Furthermore, on-the-fly score computation allows for computing scores only as needed during the local search steps. This results in obtaining good solutions without needing to enforce strong treewidth bounds typical to in-exact approaches.

ACKNOWLEDGEMENTS

This work has been financially supported by Academy of Finland (grants 276412, 312662, 322869).

REFERENCES

[1] Haley J. Abel and Alun Thomas, 'Accuracy and computational efficiency of a graphical modeling approach to linkage disequilibrium estimation', *Stat. Appl. Genet. Mol. Biol.*, **143**(10.1), (2017).
[2] Francis R. Bach and Michael I. Jordan, 'Thin junction trees', in *NIPS*, pp. 569–576. MIT Press, (2001).
[3] Mark Bartlett and James Cussens, 'Integer linear programming for the Bayesian network structure learning problem', *Artif. Intell.*, **244**, 258–271, (2017).
[4] Jeremias Berg, Matti Järvisalo, and Brandon M. Malone, 'Learning optimal bounded treewidth Bayesian networks via maximum satisfiability', in *AISTATS*, volume 33 of *JMLR W&CP*, pp. 86–95. JMLR.org, (2014).
[5] Anton Chechetka and Carlos Guestrin, 'Efficient principled learning of thin junction trees', in *NIPS*, pp. 273–280. Curran Associates, Inc., (2007).
[6] David Maxwell Chickering, 'Learning equivalence classes of Bayesian-network structures', *J. Mach. Learn. Res.*, **2**, 445–498, (2002).
[7] C. K. Chow and C. N. Liu, 'Approximating discrete probability distributions with dependence trees', *IEEE T. Inform. Theory*, **14**(3), 462–467, (1968).
[8] Gregory F. Cooper, 'The computational complexity of probabilistic inference using Bayesian belief networks', *Artif. Intell.*, **42**(2-3), 393–405, (1990).
[9] Jukka Corander, Tomi Janhunen, Jussi Rintanen, Henrik J. Nyman, and Johan Pensar, 'Learning chordal Markov networks by constraint satisfaction', in *NIPS*, pp. 1349–1357, (2013).
[10] A. Philip Dawid and Steffen L. Lauritzen, 'Hyper Markov laws in the statistical analysis of decomposable graphical models', *Ann. Stat.*, **21**(3), 1272–1317, (09 1993).
[11] Petros Dellaportas and Jonathan J. Forster, 'Markov chain Monte Carlo model determination for hierarchical and graphical log-linear models', *Biometrika*, **86**(3), 615–633, (1999).
[12] Amol Deshpande, Minos N. Garofalakis, and Michael I. Jordan, 'Efficient stepwise selection in decomposable models', in *UAI*, pp. 128–135. Morgan Kaufmann, (2001).
[13] Vibhav Gogate, William Austin Webb, and Pedro M. Domingos, 'Learning efficient Markov networks', in *NIPS*, pp. 748–756. Curran Associates, Inc., (2010).
[14] Tomi Janhunen, Martin Gebser, Jussi Rintanen, Henrik Nyman, Johan Pensar, and Jukka Corander, 'Learning discrete decomposable graphical models via constraint optimization', *Stat. Comput.*, **27**(1), 115–130, (2017).
[15] Kustaa Kangas, Mikko Koivisto, and Teppo M. Niinimäki, 'Learning chordal Markov networks by dynamic programming', in *NIPS*, pp. 2357–2365, (2014).

Table 3. Comparison of lsmarkov's on-the-fly score computation ("on-the-fly") to the precomputation of clique scores up to a given clique size cs ("precompute"). Highest scores and among those the shortest time are in bold.

		On-the-fly		**Precompute $cs \leq 3$**		**Precompute $cs \leq 4$**		**Precompute $cs \leq 5$**	
Name	n	**Score**	**Time (s)**	**Score**	**Time (s)**	**Score**	**Time (s)**	**Score**	**Time (s)**
autos	26	**-1513.2**	**100**	-1659.5	178	-1577.2	54	-1547.6	413
water10000	26	-128910.9	769	-129005.6	751	-128910.9	172	**-128907.8**	**206**
insurance10000	27	-134175.5	290	-135721.9	117	**-134154.9**	**42**	-134194.8	615
steel	28	**-18604.4**	**425**	-20203.1	324	-19358.9	353	-18898.3	297
horse	28	-4503.0	513	-4514.2	808	-4503.0	88	**-4502.9**	**16**
flag	29	-2720.0	339	-2753.9	508	-2722.1	609	**-2719.9**	**78**
Epigenetics	30	**-177782.8**	**854**	-191603.9	200	-183879.5	381	-180640.9	881
wdbc	31	**-6654.7**	**602**	-6838.1	443	-6698.1	32	-6655.4	746
mildew10000	35	-454360.8	545	-454368.7	513	**-454331.1**	**451**	-454360.8	877
soybean	36	**-2932.2**	**715**	-3405.3	544	-3254.3	24	-3135.9	146
alarm10000	37	**-105155.3**	**408**	-107446.5	564	-105178.0	715	-105161.0	424
bands	39	**-5099.6**	**350**	-5156.6	598	-5120.1	574	-5101.1	47
connect-4	43	**-1008451.1**	**171**	-1051217.1	122	-1027462.9	95	-1018349.5	711
spectf	45	**-8055.0**	355	-8057.4	202	**-8055.0**	**290**	**-8055.0**	337
sponge	45	**-1788.1**	**89**	-1850.4	792	-1815.3	16	-1815.2	21
barley5000	48	-266863.3	537	-275546.5	254	**-266736.1**	**540**	No	-
hailfinder10000	56	-497216.1	758	-498168.5	279	**-497132.5**	**453**	No	-
lung-cancer	57	-1248.9	262	-1268.3	48	-1253.8	36	**-1247.2**	**339**
promoters	58	**-8326.5**	**3**	**-8326.5**	**3**	**-8326.5**	10	**-8326.5**	107
carpo10000	60	**-173880.7**	**765**	-178730.6	460	-176580.9	86	-176166.6	849
kddcup	60	**-72971.4**	**207**	-74722.9	9	-73641.7	143	No	-
dota2	115	**-442078.8**	**20**	**-442078.8**	43	No	-	No	-
msweb	235	**-51596.0**	**881**	-52661.2	267	No	-	No	-

[16] Kustaa Kangas, Teppo Niinimäki, and Mikko Koivisto, 'Averaging of decomposable graphs by dynamic programming and sampling', in *UAI*, pp. 415–424. AUAI Press, (2015).

[17] David R. Karger and Nathan Srebro, 'Learning Markov networks: maximum bounded tree-width graphs', in *SODA*, pp. 392–401. ACM/SIAM, (2001).

[18] K. S. Sesh Kumar and Francis R. Bach, 'Convex relaxations for learning bounded-treewidth decomposable graphs', in *ICML*, volume 28 of *JMLR W&CP*, pp. 525–533. JMLR.org, (2013).

[19] Johan Kwisthout, Hans L. Bodlaender, and Linda C. van der Gaag, 'The necessity of bounded treewidth for efficient inference in Bayesian networks', in *ECAI*, volume 215 of *FAIA*, pp. 237–242. IOS Press, (2010).

[20] Steffen L. Lauritzen and David J. Spiegelhalter, 'Local computations with probabilities on graphical structures and their application to expert systems', in *Readings in Uncertain Reasoning*, 415–448, Morgan Kaufmann Publishers Inc., (1990).

[21] Colin Lee and Peter van Beek, 'Metaheuristics for score-and-search Bayesian network structure learning', in *Canadian AI*, volume 10233 of *LNCS*, pp. 129–141, (2017).

[22] Gérard Letac and Hélène Massam, 'Wishart distributions for decomposable graphs', *Ann. Stat.*, **35**(3), 1278–1323, (2007).

[23] Brandon M. Malone and Changhe Yuan, 'A depth-first branch and bound algorithm for learning optimal Bayesian networks', in *GKR 2013 Revised Selected Papers*, volume 8323 of *LNCS*, pp. 111–122. Springer, (2014).

[24] Francesco M. Malvestuto, 'Approximating discrete probability distributions with decomposable models', *IEEE T. Syst., Man, Cybern.*, **21**(5), 1287–1294, (1991).

[25] Andrew W. Moore and Mary S. Lee, 'Cached sufficient statistics for efficient machine learning with large datasets', *J. Artif. Intell. Res.*, **8**, 67–91, (1998).

[26] Mukund Narasimhan and Jeff A. Bilmes, 'PAC-learning bounded treewidth graphical models', in *UAI*, eds., David Maxwell Chickering and Joseph Y. Halpern, pp. 410–417. AUAI Press, (2004).

[27] Pekka Parviainen, Hossein Shahrabi Farahani, and Jens Lagergren, 'Learning bounded tree-width Bayesian networks using integer linear programming', in *AISTATS*, volume 33 of *JMLR W&CP*, pp. 751–759. JMLR.org, (2014).

[28] Aritz Pérez, Christian Blum, and José Antonio Lozano, 'Learning maximum weighted (k+1)-order decomposable graphs by integer linear programming', in *PGM*, volume 8754 of *LNCS*, pp. 396–408. Springer, (2014).

[29] Stephen Della Pietra, Vincent J. Della Pietra, and John D. Lafferty, 'Inducing features of random fields', *IEEE T. Pattern Anal. Mach. Intell.*, **19**(4), 380–393, (1997).

[30] Kari Rantanen, Antti Hyttinen, and Matti Järvisalo, 'Learning chordal Markov networks via branch and bound', in *NIPS*, pp. 1845–1855, (2017).

[31] Neil Robertson and Paul D. Seymour, 'Graph minors. II. algorithmic aspects of tree-width', *J. Algorithms*, **7**(3), 309–322, (1986).

[32] D. Rose, R. Tarjan, and G. Lueker, 'Algorithmic aspects of vertex elimination on graphs', *SIAM J. Comput.*, **5**(2), 266–283, (1976).

[33] Mauro Scanagatta, Giorgio Corani, Cassio Polpo de Campos, and Marco Zaffalon, 'Learning treewidth-bounded Bayesian networks with thousands of variables', in *NIPS*, pp. 1462–1470, (2016).

[34] Dafna Shahaf and Carlos Guestrin, 'Learning thin junction trees via graph cuts', in *AISTATS*, volume 5 of *JMLR W&CP*, pp. 113–120. JMLR.org, (2009).

[35] Nathan Srebro, 'Maximum likelihood bounded tree-width Markov networks', *Artif. Intell.*, **143**(1), 123 – 138, (2003).

[36] Milan Studený and James Cussens, 'Towards using the chordal graph polytope in learning decomposable models', *Int. J. Approx. Reason.*, **88**, 259–281, (2017).

[37] Tamás Szántai and Edith Kovács, 'Discovering a junction tree behind a Markov network by a greedy algorithm', *Optim. Eng.*, **14**(4), 503–518, (2013).

[38] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis, 'The max-min hill-climbing Bayesian network structure learning algorithm', *Mach. Learn.*, **65**(1), 31–78, (2006).

[39] Peter van Beek and Hella-Franziska Hoffmann, 'Machine learning of Bayesian networks using constraint programming', in *CP*, volume 9255 of *LNCS*, pp. 429–445. Springer, (2015).

[40] Claudio J. Verzilli, Nigel Stallard, and John C. Whittaker, 'Bayesian graphical models for genomewide association studies', *Am. J. Hum. Genet.*, **79**(1), 100–112, (2006).

[41] Ami Wiesel, Yonina C. Eldar, and Alfred O. Hero III, 'Covariance estimation in decomposable Gaussian graphical models', *IEEE T. Signal Proces.*, **58**(3), 1482–1492, (2010).