Maximum Likelihood Markov Networks: An Algorithmic Approach

by

Nathan Srebro

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

October 2000

Author . . . .
Department of Electrical Engineering and Computer Science
October 31, 2000

Certified by . . . .
David Karger
Associate Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Maximum Likelihood Markov Networks: An Algorithmic Approach

by

Nathan Srebro

Submitted to the Department of Electrical Engineering and Computer Science
on October 31, 2000, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

We show how a graphical model learning problem can be presented as a purely combinatorial problem. This allows us to analyze the computational hardness of the learning problem, and devise global optimization algorithms with proven performance guarantees.

Markov networks are a class of graphical models that use an undirected graph to capture dependency information among random variables. Of particular interest are Markov networks over low treewidth graphs, under which many operations are tractable. We study the problem of finding a maximum likelihood distribution among Markov networks over graphs of bounded treewidth.

We define the maximum hypertree problem, which is the problem of finding an acyclic hypergraph of bounded width, that maximizes the weight of hyperedges it covers (or equivalently, a triangulated graph of bounded clique size maximizing the weight of its cliques). We show that the maximum likelihood Markov network problem can be formulated as a maximum hypertree problem, and in fact the two problems are equivalent. This extends the work of Chow and Liu (1968) who considered the case where $k = 1$ (i.e. trees).

We show that the maximum hypertree problem is NP-hard even for $k = 2$ and give the first constant factor approximation algorithm for it. More precisely, for any fixed treewidth objective k, we find a k-hypertree with an $f(k)$ fraction of the maximum possible weight of any k-hypertree graph.

Thesis Supervisor: David Karger
Title: Associate Professor

Acknowledgments

Work on the maximum likelihood Markov network problem originated in a class taught by Tommi Jaakkola. Tommi provided continued advice and assistance and gave me insight into graphical models and the problems discussed in this thesis. I am thankful to him for his help with understanding these problem and preparing this thesis.

The combinatorial and algorithmic work presented in the thesis was done jointly with David Karger. David suggested many of the ideas presented in the thesis, and together we wrote the paper on which several chapters are based. I thank him for working with me on this problem and for supervising this thesis.

I would like to thank Jon Feldman, Nir Friedman, Daphne Koller, Marina Meila, Chris Meek, Ron Rivest, Madhu Sudan, Martin Wainwrigh and Alan Willsky and many others for helpful conversations. I would also like to thank Amos Beimel, Dana Pe’er, and especially Keren Livescu, for extensive comments on the manuscript and help in editing the thesis.

And special thanks to Elizabeth, without which I would not have survived this long process.

Contents

Chapter 1

Introduction

In this thesis, we demonstrate a rigorous combinatorial and algorithmic treatment of a machine learning problem. The machine learning problem we are concerned with is learning a maximum likelihood Markov network of bounded "complexity" (specifically, bounded *treewidth*), from an empirical sample. We show how this problem corresponds to a combinatorial optimization problem on hypergraphs, which we formulate as a "maximum hypertree" problem. We establish that the problems are equivalent by bidirectional reductions, i.e. from the maximum likelihood problem to the maximum hypertree problem and vice versa. We then use the maximum hypertree problem to prove the hardness of the maximum likelihood problem, and provide a constant-factor (for fixed "width") approximation algorithm for it.

In 1968, Chow and Liu [CL68], provided such an analysis for the limited case in which the Markov network is restricted to trees. To the best of our knowledge, this is the first generalization of such treatment to the more general case. It allows us, for the first time, to provide hardness results and provable approximation algorithms for the learning problem. The approximation algorithm is of "global" nature, solving and rounding a linear problem, as opposed to local search heuristics which have been suggested before [Mal91].

The presentation here is also, as far as we know, the first formulization of the maximum hypertree problem. In this problem, given some target width k and a weight function on candidate cliques of size up to $k + 1$, one seeks a *treewidth* k graph (i.e. a triangulated graph with maximum clique size $k + 1$) that maximizes the total weight on its cliques. The problem of finding the treewidth of a graph (and its associated *tree decomposition* or triangulation) has been extensively studies. Finding the treewidth, and tree decomposition and triangulation, of a graph is a *supergraph* problem— we seek to find a triangulated graph containing our desired graph. However, the maximum hypertree problem can be viewed as a *subgraph* problem.

The approximation algorithm we present is an initial step to providing good algorithms for learning maximum likelihood Markov networks. We hope that further study of the combinatorial problem we present will yield better algorithms. Such algorithms could then be applied to the learning problem.

We hope this thesis will be of interest both the machine learning and to the algorithms communities. We aim to give enough background so as the thesis will be approachable to readers of both disciplines.

1.1 The Learning Problem

We briefly outline the maximum likelihood Markov network problem. A more complete description is given in Chapter 2.

One of the important areas of machine learning is the development and use of *probabilistic models* for classification and prediction. One popular probabilistic model is the *Markov network*, which uses a graph to represent dependencies among the variables in the probabilistic model. Given the graph, a probability distribution on the variables can be succinctly represented by tables (called potential functions) of possible outcomes for each set of variables that forms a clique.

In order to avoid over-fitting the model, it is important that the model's graph have no large cliques. At the same time, for efficient use of the model, the graph needs to be triangulated, i.e. have no minimal cycles of more than three vertices. Combining these two objectives yields the actual requirement: that the underlying graph have small *treewidth*.

Treewidth will be defined formally later; for now we note that only trees have treewidth one, while a small treewidth means that the graph is quite like a tree. Treewidth is closely related to triangulated graphs: in a triangulated graph the treewidth is equal to the maximum clique size minus one. More generally the treewidth of a graph is the minimum over all triangulations of it, of the maximum clique size in the triangulation, minus one.

In some applications, the graphical model is specified in advance. But in others, the goal is to generate a graphical model that "best fits" some observed data (samples from an unknown distribution). Chow and Liu [CL68] show how the best *treewidth 1* model (that is, tree) for the data can be found via a maximum spanning tree computation on a graph whose weights are determined by the values of the observed data. But sometimes a higher treewidth is needed to get a good fit to the data.

1.1.1 Our contribution

We consider the more general problem: to learn, given some observed data, the maximum likelihood *treewidth* k Markov network of the data. This is the maximum likelihood triangulated Markov network with clique size at most $k + 1$.

As with the simpler case, we show how to reduce this problem to a pure graph problem. But unlike the simple case, weights on edges are no longer enough. Instead, we show how to assign weights to every subset of vertices of size up to $k + 1$. These weights are a generalization of the Chow and Liu weights and capture the information in beyond-pairwise interactions. We formulate a combinatorial problem using these weights, and through it show that:

- Finding a maximum likelihood Markov network of bounded treewidth (and so also triangulated network of bounded clique size) is NP-hard.

- For any fixed k, a Markov network of treewidth at most k (a triangulated network of clique size at most $k + 1$) can be found such that the gain in log likelihood versus an fully independent model is within a constant multiplicative factor to the maximum possible gain in log likelihood.

1.1.2 Projections

A maximal likelihood distribution is a distribution minimizing the information divergence to the empirical distribution. Finding a maximum likelihood distribution can thus be seen as an instance of the more general problem of *projecting* a target distribution onto a distribution class, i.e. finding the distribution from within the class that minimizes the information divergence to the target. Such projections have applications beyond finding the maximal likelihood distribution.

Throughout the thesis, we discuss such distribution projections, and work with this framework.

1.1.3 Related work

The problem of finding a maximum likelihood Markov network of bounded tree width has been investigated before and discussed in [Pea97]. Malvestuto [Mal91] discussed the connection between this problem and maximal acyclic hypergraphs (which we call *hypertrees* here), and suggested a local search heuristic on hypertrees.

Several other extensions to the work of Chow and Liu [CL68] for tree-shaped Markov networks have recently been proposed. Meila [MP99] suggested modeling distributions as mixtures of tree-shaped Markov networks. Dasgupta [Das99] suggested polytree Bayesian networks (trees with oriented edges).

There is also work on *directed* graphical models known as *Bayes Networks*. Dagum and Luby have results that focus on the problem of, given a specific graphical model, learning the appropriate setting of the joint probability distributions. They show that even achieving good approximations for this problem is NP-hard in the general case [DL93], but also give approximation algorithms that work well on a large class of instances [DL97].

1.2 The Algorithmic Problem

Given a candidate graph with weights on edges, and also on larger cliques of size up to $k+1$, we would like to find the maximum weight treewidth-k subgraph of the input graph. For $k > 1$, this problem is NP-complete. We develop approximation algorithms for it. For an n-vertex graph with goal width k, in time $n^{O(k)}$, we find a treewidth-k graph containing at least an $f(k)$ fraction of the maximum possible weight.

The running time of our algorithm is unsurprising, since the input problem size can (and will often in practice) be $n^{O(k)}$: a weight may be need to be specified for every clique of size up to k. It is not clear whether the dependence of our approximation factor on the goal treewidth k is necessary, but we do in any case get a (weak) constant factor approximation for every fixed k, which is the case that is dealt with in practice.

Our approximation algorithm is based on two main observations. The first is the identification of a structure called a k-*windmill*. While treewidth-k graphs can have quite a complicated structure, k-windmills are easier to work with. We show that any treewidth-k graph places at least a constant fraction of its weight in disjoint k-windmills, and thus settle for approximating maximum weight disjoint k-windmills. To find these windmill, we develop a linear-programming-based approximation algorithm. The linear program bears some faint resemblance to those in recent algorithms for *facility location* [STA97]. Our rounding scheme is quite different, however, and has an interesting "iterative" approach similar to Jain's algorithm for network design [Jai98]: after solving the LP, we randomly round *some* of the fractional variables; we then *re-solve* the linear program to make it feasible again before we proceed to round other variables.

Treewidth has been defined in many different contexts and using various equivalent definitions. We present some of these in Chapter 3, but the setting we use throughout the thesis is that of acyclic hypergraphs.

1.2.1 Related work

Finding maximum-weight subgraphs meeting some property is of course a broad field; a recent good example is the maximum planar subgraph work of Călinescu et al. [CFFK98].

Most recent work on treewidth has been concerned with showing, given some input graph, that the graph *has* small treewidth, and on finding an appropriate tree decomposition [SG97, Bod97, Bod96]. Here, we focus on a different problem. We would like to find a graph of treewidth at most k that captures the greatest weight. We do not expect to be able to include all the edges of the graph, but rather aim to maximize what can be included. While finding a tree-decomposition of a given graph might be viewed as a covering problem (finding a low-treewidth graph containing the target graph), our problem is a sub-graph problem—finding a maximal small-treewidth graph inside a given graph.

1.3 Structure of the Thesis

This thesis is contains two main threads: a purely combinatorial analysis of a combinatorial optimization problem (the maximum hypertree problem), and an analysis of the equivalence between a learning problem (maximum likelihood, or projected, Markov networks) and the combinatorial problem, and the consequences of this equivalence.

The rest of this thesis is structured as follows:

- In Chapter 2 we introduce the notions of a maximum likelihood Markov network and Markov network projections. We motivate the general learning setting and formulate the specific learning problem which we tackle in this thesis.

  The chapter serves mostly as an introduction and tutorial for readers unfamiliar with unsupervised machine learning and graphical models.

- Chapter 3 serves as a tutorial on treewidth and the related concepts of tree decompositions and acyclic hypergraphs. It provides several equivalent definitions of these concepts, and presents some known results which are used later in the thesis.

- in Chapter 4 we formally define the maximum hypertree problem, and prove its hardness. We also present some properties of hypertrees that might be of use in solving the problem, but that we do not use in this work.

- Chapter 5 is the core of the second thread, and the links between them. It presents the equivalence between the learning problem and the combinatorial problem. The first sections present known results about decompositions of Markov networks over acyclic hypergraphs (or equivalently, triangulated graphs). Sections 5.3 and 5.4 present new results, proving the bidirectional equivalence.

- Chapters 6 and 7 hold the core algorithmic content of the thesis. In Chapter 6 windmills, and the "maximum windmill forest" problem are presented, and it is shown that a maximum windmill forest serves as an approximation to the maximum hypertree. Chapter 7 presents an approximation algorithm for the maximum windmill forest problem, which translates to an approximation algorithm for the maximum hypertree problem

A reader interested only in the algorithmic thread and in the presentation of a new combinatorial optimization problem, may choose to skip Chapters 2 and 5, without loss of understanding of the combinatorial issues presented in the other chapters.

A reader interested only in the learning thread, may focus only on Chapter 2, parts of Chapter 3 and Chapter 5. The relevant implications on the machine learning problems are presented in these chapters.

Chapters 2, 3 and Sections 5.1 and 5.2 contain background material and review of known results. The results in Section 4.2, 5.3, 5.4 and Chapters 6 and 7 are new results first presented in this thesis.

Some of the results presented in this thesis are to be published in [KS01].

Chapter 2

Introduction to Density Estimation, Distribution Projections, Maximum Likelihood, and Markov Networks

In this chapter we introduce and motivate the notion of a maximum likelihood Markov network and Markov network projections, and formulate the learning problem which we tackle in this work.

The chapter is intended mostly for readers unfamiliar with unsupervised machine learning and with graphical models. It provides all the necessary background about the underlying machine learning issues, the motivation for the algorithmic problem, and for understanding the rest of this manuscript, particularly Chapter 5. It can, however, be skipped, together with Chapter 5, without loss of understanding of the combinatorial and algorithmic details in the other chapters.

The chapter also serves to set the basic framework, and clarify the learning scenario addressed by this work, emphasizing the differences from other unsupervised machine learning problems.

Some well known properties of Markov networks are presented in this chapter without proof. Most are proved, sometimes in a different formulation, in Chapter 5.

2.1 The Machine Learning Setting: Density Estimation and Maximum Likelihood Distributions

2.1.1 Density estimation

One of the challenges of unsupervised learning, given a sample of observations, is to determine the distribution law from which the samples were drawn. The predicted distribution can be used to make predictions about future, partially observed data. Often, each observed data point is taken to be expressed as a vector of variables $x = (x_1, \ldots, x_n)$. A common approach in probabilistic machine learning is to assume each data vector is drawn independently at random from the same unknown probability distribution P^0 over possible vector values. One then aims to *learn* P^0 from the samples.

We will use the following notation: Random variables are generally represented by uppercase letters, and their outcomes by lower case letters. We denote by n the number of random variables in a single sample: P^0 is a distribution over random vectors of length n. We denote by T the number of observed samples, $x^1, \ldots, x^T$, where $x^t = (x^t_1, \ldots, x^t_n)$. We assume each $X^t \sim P^0$ independently. Note that the variables $X^t_1, \ldots, X^t_n$ within a single sample vector X^t are *not* necessarily independent, but the sampled vectors X^t are independent of each other. Based on the observations $X^t = x^t$, we would like to estimate P^0. That is, we would like to learn a distribution $P^\sharp$, such that $P^\sharp$ is "close" to P^0.

By "close" we mean that P^0 and $P^\sharp$ assign similar probabilities to events. This can be quantified by various measures, the most natural of which is perhaps the information divergence $H\left(P^0 \| P^\sharp\right) = \mathbf{E}_{P^0}\left[\log \frac{P^0}{P^\sharp}\right]$.

The empirical distribution and overfitting

One possible candidate for $P^\sharp$ is the empirical distribution[1] of the samples, $\hat{P}$. However, this is usually a very bad choice as $\hat{P}$ will grossly overfit the data and will not generalize

[1]The distribution which assigns to each outcome its frequency in the observed samples

well to unobserved outcomes. In most scenarios, especially when the dimension n is large, it is not likely that every possible outcome vector x will be encountered, as the number of possible outcome vectors is exponential in n. But $\hat{P}$ associates a probability of zero to any unencountered outcome, concentrating too much on the encountered outcomes, which are usually but a small sample of all possible outcomes.

Without making any assumptions, or speculations, about the nature of the distribution P^0, not much further can be done— if we assume nothing about the behavior of P^0 on different outcomes, there is no way to generalize from the observed values to yet unobserved ones. In order to make such generalizations, we must use prior knowledge, speculations, or assumptions, about P^0, e.g. that it is smooth in some way, that similar values are related, or that it has only limited internal dependencies.

Limiting the distribution to prevent overfitting

A possible approach is to choose a distribution from within a limited class of distributions $\mathcal{D}$. This limited class represents our prior assumptions, or speculations, about the true distribution P^0, or its properties.

Focusing on a specific class $\mathcal{D}$, a reasonable choice it to choose the distribution $P^\sharp \in \mathcal{D}$ which maximizes the probability of observing the data:

$$P^\sharp = \arg\max_{P \in \mathcal{D}} P(X^1 = x^1, \ldots, X^T = x^t) \tag{2.1}$$

The distribution $P^\sharp$ is called the *maximum likelihood distribution*, where the *likelihood* of a distribution is the probability of observing the data under that distribution.

Note that the maximum likelihood distribution is also the distribution from within $\mathcal{D}$

that minimizes the information divergence with $\hat{P}$:

$$\begin{aligned}
P^\sharp &= \arg\max_{P\in\mathcal{D}} P(X^1 = x^1, \ldots, X^T = x^t) \\
&= \arg\max_{P\in\mathcal{D}} \prod_t P(x^t) \qquad\qquad X^i \text{ are independent} \\
&= \arg\max_{P\in\mathcal{D}} \sum_t \log P(x^t)
\end{aligned}$$

replacing summation over observed outcomes with a sum over all possible outcomes, counting the number of times they were observed using the empirical distribution,

$$\begin{aligned}
&= \arg\max_{P\in\mathcal{D}} \sum_x T\hat{P}(x)\log P(x) \\
&= \arg\min_{P\in\mathcal{D}} -\sum_x \hat{P}(x)\log P(x)
\end{aligned}$$

The distribution $\hat{P}$, and so also any function of it, is constant, and adding it does not change the minimizing distribution:

$$\begin{aligned}
&= \arg\min_{P\in\mathcal{D}} \left(\sum_x \hat{P}(x)\log\hat{P}(x) - \sum_x \hat{P}(x)\log P(x^t)\right) \\
&= \arg\min_{P\in\mathcal{D}} \sum_x \hat{P}(x)\log\frac{\hat{P}(x)}{P(x)} \\
&= \arg\min_{P\in\mathcal{D}} H\left(\hat{P}\|P\right) \qquad (2.2) \\
& \qquad (2.3)
\end{aligned}$$

Since the information divergence can be viewed as a “distance” measure[2], we refer to $P^\sharp$ as the *projection* of $\hat{P}$ onto $\mathcal{D}$. More generally a *projection* of a some target distribution (not necessarily an empirical distribution of some sample) onto a class of distributions, is the distribution from with in the class minimizing the information divergence to the target.

[2]Although it is not a metric.

2.1.2 Estimation error and generalization error

Limiting to a restricted class $\mathcal{D}$ can reduce the risk of overfitting. For example, we might limit to the class $\mathcal{D}_0$ of distributions in which the variables X_i in the observed vector are independent. In this case, to estimate the maximum likelihood distribution $P_0^\sharp \in \mathcal{D}$, one need only estimate the marginals over each variable separately. Since these marginals have only a few possible outcomes (compared to the total number of outcome combinations), a good estimate can be attained with a relatively small number of samples.

However, if there are significant dependencies between variables in the true distribution P^0, as may well be the case, then $P_0^\sharp$ will not approximate P^0 well, because no distribution in $\mathcal{D}_0$ approximates P^0 well.

We distinguish here between two sources of "error", i.e. discrepancies between the estimated distribution $P^\sharp$ and the true distribution P^0:

The approximation error is the discrepancy between P^0 and the class $\mathcal{D}$, i.e. the difference between P^0 and the distribution $P^* \in \mathcal{D}$ that is closest to it.

The estimation error is the difference between P^* and our estimate of it based on the observed samples, $P^\sharp$.

The estimation error is essentially caused by not having enough samples. Had we an infinite number of samples (and infinite time), we could find P^* exactly. The fewer samples we have, the greater the estimation error is likely to be. The estimation error also depends on the size of the class $\mathcal{D}$: intuitively, the smaller, and simpler, the class, the easier it is to "focus in" on P^*, and fewer samples will be needed to reduce the estimation error.

The approximation error does not depend on the number of samples, but only on the distribution class $\mathcal{D}$. The bigger the class $\mathcal{D}$, and denser it is in the space of all distributions, the more conceivable it is that there will be a distribution $P^* \in \mathcal{D}$ that will approximate P^0 well. Of course, not only the size is important, but perhaps more important is choosing a class $\mathcal{D}$ which correctly captures the expected properties of P^0, and so ensures that P^0 will lie within the class, or at least not far from it.

We see a tradeoff between the estimation error and the approximation error, controlled by the size of the class $\mathcal{D}$. A bigger and more complex $\mathcal{D}$ might capture P^0 better and reduce the approximation error, but at the same time make it harder to estimate P^* and increase the estimation error. We would like to choose a distribution class that is simple and small enough to be estimated using the samples we have, yet as large and comprehensive as possible to allow for a good approximation of P^0. The more samples we have, the better we can estimate even in more complex classes, and the larger the class we will aim to use.

2.1.3 Limiting to Markov networks

Earlier we described one possible, rather simple, distribution class— the class $\mathcal{D}_0$ of distributions with no dependencies between random variables. Finding the maximum likelihood distribution from this class is straightforward. But the class is very limited and often one would like to allow some dependencies in the distribution.

We might like to use larger, more complex, classes, when we have enough samples to support estimation in those classes. It is convenient to use a parameterized family of distribution classes, which gradually become larger and more complex. We can then use the distribution class from the family that is appropriate for the sample size at hand.

A possible more general family of distributions are *Markov networks*, which will be described in detail in Section 2.2. Markov networks allow a limited dependency structure, as imposed by a graph (loosely speaking, dependencies are only allowed along edges in the graph). The denser and "wider" the graph, the less restricted the distribution. In this work, we consider the problem of finding a maximum likelihood distribution from within the class $\mathcal{D}_k$ of Markov networks of width at most k (as will be defined in Chapter 3).

2.1.4 Density estimation, not model selection

It is important to note that the problem we concentrate on is density estimation, and not model selection or hypothesis testing. We discuss model selection problems briefly, in order to emphasize what we do *not* do.

In model selection problems, we aim to discover the underlying distribution model. For example we might want to decide which random variables are independent. Since a more complex model (e.g. with more dependencies) will always predict the data better, and have a higher likelihood, a pure maximum likelihood approach is not suitable in this scenario. Instead, we wish to balance likelihood and simplicity, and find a model that is both simple (e.g. assume as few dependencies as possible) and predicts the data well.

But in this thesis we do *not* consider the problem of model selection. In the scenario we are concentrating on, we are merely trying to estimate a distribution, and our output is a distribution. The quality is measured by how well the distribution itself, i.e. the probabilities assigned to possible outcomes, resembles the true distribution. We are limiting to a class of simple distributions only to overcome overfitting— had we more samples, we would allow ourselves to choose from a wider, more complex, class of distributions, since this will always decrease (or at least, not increase) the approximation error. This is in sharp contrast to model selection, where even if we have an infinite number of samples, we would still prefer a simple model.

2.1.5 Tractable models

Despite our emphasis on the *distribution* rather then the *model*, there is one sense in which we are concerned also with the underlying model, or representation of the distribution. To be of any practical use, the resulting distribution must be representable in some compact form that allows efficient computation of marginal (and thus also conditional) probabilities. Recording the probability value associated with each possible outcome is almost always infeasible, because of the huge (exponential in n) number of possible outcomes, and calculating a marginal probability with such a verbose representation requires excessive computation.

Existence of a compact, but not necessarily tractable, representation is tied to the size of the distribution class, since the length of the minimal representation is logarithmic in the size of the class. Since we restrict the size of the class to avoid overfitting, we also implicitly restrict the size of its minimal representation. In fact, from the information theoretic point of view, the length of the representation of the distribution surely cannot be more than the sample size, since this is our only source of information about the distribution.

However, although such a compact representation is guaranteed to exist, computations using it, perhaps even point probability calculations, might be intractable. We will probably need to limit our distribution class $\mathcal{D}$ only to tractable distributions, i.e. distributions that have a representation supporting efficient marginal probability calculations. Note that this restriction is imposed as a practical necessity of being able to make efficient use of the resulting distribution, and not as part of the mathematical framework of distribution estimation.

Other than tractable computation of marginal probabilities, we might be interested in other representation or computational properties of the distribution, such as factorizability.

2.1.6 Other approaches

We consider an approach to density estimation by limiting the distribution to a class of Markov network distributions $\mathcal{D}_k$. This is of course not the only approach to density estimation.

Other families of distributions

Many of the common methods for density estimation, and its closely associated problems of regression and classification, follow a similar approach, but with different families of distribution classes. Some of the common families used are Bayes networks and mixture families, e.g. mixtures of Gaussians, or even mixtures of limited Bayes-networks or

Markov networks.

Using a prior over distributions

Instead of restricting to a class of equally permissible distributions, while totally disallowing any other distribution, one might choose to make a "softer" limitation. This can be done using a prior distribution over the possible distributions of X. We can then select the distribution of X with the highest *a posteriori* probability: the a posteriori probability is the probability of X having a certain distribution given the observed samples, i.e. the product of the likelihood of the distribution and its prior probability.

The prior distribution reflects our belief as to which models are a priori more or less likely. For example, we might assign simpler models a higher prior than more complex models.

A true Bayesian would argue that restricting to a class of distribution, and seeking the maximum likelihood in the class, is just assigning a uniform[3] prior over that class.

Learning the parameters of a specific structure

It is also common to impose a specific structure, determined beforehand by external prior knowledge about the distribution, and fit the distribution within this model structure. For example, a specific perceptron architecture may be specified, or a specific directed graph for a Bayes network.

This is also often done with Markov networks, where a specific graph is predetermined, and the most likely Markov network on it is sought. This requires extensive prior knowledge about the distribution. This problem is well studied, and discussed in Section 2.3.

[3]with respect to some parameterization

Regularization

Other approaches suggested by modern developments in statistical learning theory, aim to balance the approximation error and estimation error dynamically. Instead of pre-limiting the level of complexity and searching for the maximum likelihood distribution within those limits, a "regularization penalty", proportional to some measure of the distribution's complexity, is combined with the likelihood, seeking a model that is both likely and non-complex. These approaches are not discussed in this work.

2.2 Markov Networks

In this section we give a brief introduction to Markov Networks. We formally define this family of distributions, and describe some known results about the family. We do not prove these results here, but most are proved, in a slightly different formulation, in Chapter 5.

2.2.1 Definition

We first formally define the family of distributions we refer to as Markov Networks. In the discussion below, X is a random vector, and x is a possible outcome value for X. X_v is an element of X, i.e. a random variable corresponding to the value of X in one of its coordinates. x_v is a possible outcome of X_v.

Definition 2.1. *We write $A \perp B \mid C$ if for variable sets A, B, and C, conditioned on any values of the variables in C, the variables in A are independent of those in B.*

Definition 2.2 (Markov Network). *A random vector X_V, indexed by vertex set V, is a* Markov network *over an undirected graph*[4] *$G(V)$ iff each random variable X_v, conditioned*

[4]*A* graph *is a collection of* edges *between* vertices*. The* neighbors *of a vertex are the vertices to which it has edges. See Chapter 3 for complete definitions*

on its neighbors, is independent of all other elements of X_V:

$$\begin{aligned}&(\forall v \in V) \\ &X_v \perp \{X_u \mid u \neq v, (u,v) \notin G\} \mid \{X_u | (v,u) \in G\}\end{aligned} \tag{2.4}$$

It follows that if C separates A and B in G, then for the corresponding sets of random variables, $X_A \perp X_B | X_C$.

Every distribution is a Markov network over the fully connected graph (a graph in which every two vertices are connected by an edge), since then the independence requirement is satisfied vacuously. In a Markov network over the empty graph, all the variables are independent. As a more interesting example, any finite length Markov chain is a Markov network whose underlying graph is a path: each variable is dependent on only its predecessor and successor.

2.2.2 Hammersley-Clifford clique factorization

The Hammersley Clifford theorem characterizes the distributions which follow the Markovian independencies given by a graph, for distributions without so-called "forbidden combinations" [Bes74]:

Definition 2.3 (Strictly Positive Distribution). *A random vector X is distributed* strictly positively *iff for each vector of outcomes $x = (x_1, x_2, \ldots, x_n)$ for which each element has positive marginal probability $P(X_i = x_i) > 0$, then $P(x) > 0$. That is, the support of the distribution is a cartesian product of the supports for each element, meaning there are no forbidden combinations of values.*

Theorem 2.1 (Hammersley-Clifford Theorem). *A strictly positively distributed random vector X is a Markov network specified by $G(X)$ if and only if its distribution can be* *factored to the cliques in G:*

$$P_X(x) = \prod_{h \in \text{Clique}(()G)} \phi_h(x_h) \tag{2.5}$$

for some set of clique factors $\{\phi\}$*, such that ϕ_h is a function of the outcomes of random variables indexed by the clique $x_h = \{x_v | v \in h\}$*

For each clique h, the factor function ϕ_h assigns a value to each combination of possible outcomes of variables in the clique.

The sizes of cliques in the Markov network determines the complexity of both expressing and learning the distribution. For a given Markov network, the description of the distribution, and thus also the sample size needed to estimate it, is exponential in the clique sizes. For each clique h, we need to specify the value of its factor function ϕ_h for every possible argument. A clique on k variables, even if they are only binary variables, takes on 2^k compound values. We need to record (and to estimate) the value of ϕ_h for each of those 2^k input values.

2.2.3 Triangulated Markov networks

While equation (2.5) provides an explicit formula for using the clique factors to calculate the probability of an outcome, calculating marginal probabilities using this representation is not necessarily easy, and might require summation over all outcomes of nuisance variables. Similarly, there is no direct way of calculating the appropriate factors for a given distributions.

In a certain class of graphs, however, such calculations are possible.

Triangulated graphs are graphs with no minimal cycles of more than three nodes. They are discussed in detail in Section 3.3.4. Over such graphs, marginal, and hence also conditional, probabilities can be calculated directly from the clique factors in linear time [WL83], i.e. linear in the size of the tables used for computing the factors, and hence exponential in the clique sizes. Conversely, for triangulated G, the clique factoring can be given explicitly as a function of the marginal distributions over the cliques (proven in Theorem 5.1):

$$\phi_h(x_h) = \frac{P_h(x_h)}{\prod_{C' \subset C} \phi_{C'}(x_{C'})}. \tag{2.6}$$

Note that this representation requires a product over all cliques in (2.5), including non-maximal cliques. Factors corresponding to non-maximal cliques can of course be subsumed into some containing maximal clique factor. However this leads to clique factors which are dependent on the graph structure. The factors given by (2.6) are unique in that a clique's factor does not depend on the graph G, except the fact that it include the clique.

Local dependence on marginals Other than the efficient and explicit calculations, it is also important to note that the dependence between the clique factors and the marginal distributions is *local*. That is, a clique factor depends *only* on the marginal distribution of the clique, and the marginal distribution of a clique depends *only* on factors of the clique and its sub-cliques. This is contrary to the non-triangulated case in which a change in a marginal distribution can propagate to factors of far away cliques, and visa versa.

If G is triangulated, the Hammersley Clifford theorem holds for any distribution, including distributions with forbidden combinations. This will be shown in Section 5.1.

The explicit factoring also allows for simple calculation of the maximum likelihood Markov network over a specified triangulated graph. Following (2.6), it can be shown (see Corollary 5.3) that the maximum likelihood Markov network over a given graph structure G is given by:

$$\hat{\phi}_h(x_h) = \frac{\hat{P}_h(x_h)}{\prod_{C' \subset C} \hat{\phi}_{C'}(x_{C'})} \tag{2.7}$$

Where $\hat{P}_h$ are the empirical marginal distributions over the cliques—that is, the fraction of the observed data points that took on given values.

2.3 Maximum Likelihood Markov Networks

When the triangulated graph is specified in advance, (2.7) makes it simple to assign the maximum likelihood factor functions. In some cases, however, the structure of the dependency graph is unknown, and we would like to determine both the best graph *and* the best parameter settings for it based on the empirical data. This is the main problem with which we are concerned: find a graph and a Markov network over the graph, which maximizes the likelihood of the data.

The complete graph always has maximum likelihood

Lack of edges in the graph represent independencies which must hold. Thus, adding edges to a graph relaxes the constraints on Markov networks defined by it. If X is a Markov network over graph G, then it is also a Markov network over a supergraph $G' \supset G$, and in particular also a Markov network over the fully connected graph (in which every two vertices have an edge between them). In fact, *every* random vector X is a Markov network over the fully connected graph. And so, the empirical distribution of the data, which is always the maximum likelihood distribution, is a Markov network over the fully connected graph. Thus, the fully connected graph can always be used to maximize the likelihood.

In most cases the fully connected graph will be the only graph which achieves the maximum likelihood. Even if the real distribution from which the data is sampled is a Markov network over a sparser graph G, the empirical distribution will almost surely[5] deviate slightly from the true distribution, and will *not* be a Markov network over G.

Limiting the space of admissible models

As discussed in Section 2.1.1, the empirical distribution is in most cases a vast overfitting of the data.

[5]Strictly speaking, for a continuous distribution, with probability one it will not be a Markov network over G. If the distribution is not constant, then as the number of samples increases, the probability of the empirical distribution being a Markov network will go to zero

Instead, we would like to limit the space of admissible models, as represented by the number of parameters allowed. As discussed above, the number of parameters is essentially exponential in the clique sizes. We would thus like to limit the sizes of the cliques in the graph.

A simple way of doing so is bounding the maximum clique size of the graph. We will choose a clique size bound[6] $k+1$. and search for a maximum likelihood distribution among those which are Markov networks over a graph where all cliques are of size at most $k+1$.

Bounding the clique size bounds the number of parameters, however it is not equivalent to bounding the number of parameters. A graph that contains a single clique of $k+2$ nodes, and no other edges would not be admissible. However, a graph which contains many cliques of size $k+1$ might have more parameters (if the number of values a variable can take is low, e.g. if all variables are binary).

In many ways, it might be more "correct" to bound the actual number of parameters, and not the maximum clique size, in order to allow for such non-uniform graphs. This would roughly mean bounding the sum of exponents of the clique sizes.

However, the uniform requirement of a bounded clique size yields substantially simpler combinatorial properties, and is independent of the number of possible outcomes for each random variable.

Limiting only to tractable models

For a non-triangulated graph, even if the graph structure is known, finding the maximum likelihood parameters is hard. It is conceivable that finding the maximum likelihood structure *and* the parameters is easier, especially if it is true that the maximum likelihood graph always has some restricted structure. However, we do not know that this is the case, and so we cannot expect that finding the maximum likelihood structure and parameters will be easier. Additionally, a triangulated model is more useful as a predictive model since calculations (e.g. conditional and marginal probabilities) on it are feasible (linear in the number

[6]The choice of $k+1$ and not k will be motivated in Section 3.

of parameters), whereas in a general graph they are difficult, as discussed before.

Because of this, we choose to limit the acceptable models to only triangulated graphs with cliques size at most some bound $k+1$.

Triangulated, bounded clique size, graphs also have a more "regular" number of parameters then general bounded clique-size graphs. The number of cliques of size $k+1$ is at most $n-k$ and the number of parameters is at most $m^k\left((n-k)(m-1)+1\right)-1$ (where n is the number of variables and m is the number of possible outcomes for each variable[7], and both of these bounds are attained by every maximal graph of the family. This regularity provides for a better approximation to our underlying desire to directly bound the number of parameters.

2.3.1 Summary: our learning goal

Our goal is thus to find a maximum likelihood Markov network over a triangulated graph G with clique size at most some bound $k+1$. As will be discussed later, this is equivalent to requiring the graph have *tree width* at most k.

This is an extension of the work of Chow and Liu [CL68], which showed how to find the maximum likelihood Markov network over a tree. A tree is a triangulated graph with clique size at most 2, and so the Chow Liu algorithm solves the above problem for $k=1$.

2.3.2 The learning goal as a projection problem

As was shown in Section 2.1.1, the maximum likelihood distribution in a distribution class is the projection of the empirical distribution onto the class. Thus, we can view the problem of finding a maximum likelihood distribution as a projection problem. In this thesis, we take this approach and discuss the problem of projecting a distribution onto the class of Markov networks over a triangulated graph G with clique size at most $k+1$.

[7]If the number of outcomes is not the same for all variables, m is the bound on the number of outcomes, but the bound is not attained by maximal graphs, and different maximal graphs will have a different number of parameters

This more generalized setting has applications beyond maximum likelihood estimation. We might have a target distribution that is specified in some other way, perhaps through a more complex model, as we would like to realize it as best we can using a simpler Markov network.

2.3.3 Learning a distribution versus learning a structure

One of the common uses of graphical models, including Markov networks, in unsupervised machine learning, is to understand the dependency structure of a sampled distribution. That is, given samples from an unknown distribution, learn the true, minimal, structure of its dependency graph G, such that the distribution is a Markov network over G. For example, for two variables, we would like to decide if they seem to be dependent or independent (i.e. if their dependency graph is empty, or includes the edge between them).

This type of application is a *model selection* problem, as described in Section 2.1.4, and the straightforward maximum likelihood approach is not suited for it— adding edges will always increase the likelihood.

We emphasize again that this work concerns learning a distribution, with the graphical model being a convenient way to represent the learned distribution. The methods discussed are generally not appropriate for learning the structure of a distribution.

In this sense too, this is an extension of Chow and Liu [CL68]. There too, the maximum likelihood tree is found, even though some of its edges may be superfluous.

Chapter 3

Treewidth and Hyperforests

In this chapter we introduce the concepts of *hyperforests* and the *treewidth* of a graph, and review the relevant background about them. We also formally define the maximum hypertree problem — the subject of this thesis.

All of the material in this chapter is based on previously known results. However, some of the definitions, formulations of theorems, and proofs vary from those in the cited sources.

We first recall the basic definitions of graphs and hypergraphs and present the terminology we use. In Section 3.1 we introduce hyperforests (also known as acyclic hypergraphs), hypertrees, and the related measure of the treewidth of a graph. In the rest of the chapter we review relevant known results about hyperforests and treewidth of graphs. In Section 3.2 we present some basic properties which we use throughout the thesis. In Section 3.3 we discuss several equivalent characterizations of hyperforests and treewidth. Section 3.4 points the interested reader to further work about, or using, hyperforests and treewidth.

Preliminaries: Graphs and Hypergraphs

We give the basic definitions of graphs and hypergraphs, and present the terminology used in this thesis.

A *graph* $G(V)$ is a collection of unordered pairs (*edges*) of the *vertex* set V: $G(V) \subset$

$\binom{V}{2}$. A *path* in a graph $G(V)$ is a sequence $v_1, v_2, \dots, v_r$ of distinct vertices, such that $\forall_{1 \le i < r} \{v_i, v_{i+1}\} \in G$.

A *hypergraph* $H(V)$ is a collection of subsets (edges, or sometimes explicitly *hyperedges*) of the vertex set V: $H(V) \subset 2^V$. If $h' \subset h \in H$ then the edge h' is *covered* by H. We slightly abuse set-theory notation and denote $h' \in H$ even if h' is just covered by H. A hypergraph (or graph) H' is covered by H iff $\forall_{h' \in H'} h' \in H$ (all edges in H' are covered by H, i.e. are a subset of an edge of H); if so we write $H' \subset H$. Another way of viewing this notion of a hypergraph is requiring that a hypergraph include all subsets of its edges.

A hypergraph in which all maximal edges have the same size k will be called a *k-edge-regular hypergraph*. A graph is simply a 2-edge-regular hypergraph. To emphasize the distinction between covered edges (which will have smaller size) and the maximal, regularly-sized, edges, these edges will be referred to as *regular edges*.

If a vertex is contained in (hyper)edge, the edge is said to be *incident* on the vertex. Two vertices both belong to a common (hyper)edge are said to be *adjacent*. The *neighbors* of a vertex are all vertices to which it is adjacent.

For a (hyper)graph $H(V)$ and vertex set $V' \subset V$, we denote by $H[V']$ the *induced sub-(hyper)-graph* defined by $H[V'] = \{h \cap V' | h \in H\}$. Note that for hypergraphs, the induced sub-hypergraph includes also hyperedges *covered* by the original hypergraph. For example, if $H = \{\{a, b, c\}, \{c, d\}\}$ then $H[\{a, b, d\}] = \{\{a, b\}, \{d\}\}$.

For sets V_1 and V_2, we use the notation $V_1 \setminus V_2 \doteq \{v \in V_1 | v \notin V_2\}$. For a set V and an element $v \in V$, we denote $V - v = V \setminus \{v\}$ and for an element $v \notin V$ we denote $V + v = V \cup \{v\}$.

3.1 Hyper Trees and Tree Decomposition

Hypertrees generalize trees (here referred to explicitly as 1-trees) to hypergraphs. It will be simpler to introduce hypertrees by first introducing hyperforests, which generalize forests. Recall that a forest is an acyclic graph, i.e. a graph with no cycles. The generalization of acyclicity to hypergraphs is somewhat more complex. There are several equivalent definitions for hypergraph acyclicity, which will be discussed in Section 3.3. Here, we define acyclicity using the notion of a tree structure:

Definition 3.1 (Tree Decomposition). *A hypergraph $H(V)$ is said to have* tree structure $T(H)$ *iff T is a tree over all the hyperedges of H and the following* decomposition property *holds:*

- *If $(h_1, h_2, \dots, h_k)$ is a path of H-edges in T, then $(\forall 1 < i < k)\ h_1 \cap h_k \subseteq h_i$.*

Definition 3.2 (Acyclic Hypergraph). *A hypergraph is* acyclic *iff it has a tree decomposition. An acyclic hypergraph is also referred to as a* hyperforest.

A 1-tree $H(V)$ has the following tree structure $T(H)$: if a vertex in v has degree 2 in H, then the two edges incident on it in H are neighbors in T. If a vertex v has degree higher than 2, choose one of its incident edges $h_v \in H$ arbitrarily— all other edges incident on v in H neighbor h_v in T. Note that if H is not a path, the tree structure is *not* unique, and depends on the choice of the arbitrary mapping $v \mapsto h_v$.

Two hyperforests $H_1(V_1)$ and $H_2(V_2)$, over disjoint[1] vertex sets $V_1 \cap V_2 = \emptyset$, with tree decompositions $T_1(H_1)$ and $T_2(H_2)$, can be joined to form a hyperforest $H_1 \cup H_2$, with tree decomposition $T_1 \cup T_2 \cup \{(h_1, h_2)\}$, created by adding an arc between any two arbitrary hyperedges $h_1 \in H_1$ and $h_2 \in H_2$. Thus, a 1-forest, being a union of disjoint 1-trees, is a hyperforest.

Unlike adding edges to a regular graph, adding hyperedges to a cyclic-hypergraph might make it acyclic. In fact, a hypergraph containing the complete hyperedge (the hyperedge containing all vertices) is always a hyperforest. The tree structure of such a hyperforest is a star, with the complete hyperedge in the center.

Definition 3.3 (Width of a Hyperforest). *The* width *of a hyperforest $H(V)$ is the size of the largest edge, minus one:* $\max_{h \in H} |h| - 1$.

[1]Note that it is essential that the vertex sets be disjoint. Even for 1-forests, joining two paths might create a cycle. In Section 3.2.2 we relax the disjointness restriction somewhat.

Thus, the width of a standard tree is 1. We will refer to a hyperforest of width at most k as a k-hyperforest.

Definition 3.4 (Hypertree). *A hyperforest that is maximal among hyperforests of width at most k (i.e. no $k+1$-edge can be added to it) is said to be a k-*hypertree*.*

Since we are concerned with maximum structures, we will be interested mostly in hypertrees.

Definition 3.5 (Tree Decomposition of a Graph). *A* tree decomposition *of a graph $G(V)$ is a covering hyperforest $H(V) \supseteq G(V)$ with tree structure $T(H)$.*

Recall that H covers G if every edge of G is contained in some hyperedge of H.

Definition 3.6 (Treewidth). *The* treewidth *of a graph is the width of its narrowest tree decomposition, i.e. the narrowest hyperforest covering it.*

Every graph can be covered by the hyperforest containing the complete hyperedge, so every graph has treewidth at most $n-1$.

A word about nomenclature To try to minimize confusion and ambiguity in the exposition, we will take advantage of the common parallel nomenclatures for graphs: we will use the terms *vertex* and *edge* or *hyperedge* when discussing graphs such as G and H, and reserve the terms *node* and *arc* for the tree structure of hyperforests.

3.2 Basic properties of hyperforests and tree decompositions

We present here some basic properties of hyperforests and tree decompositions, that we use throughout the thesis.

3.2.1 Cliques

Lemma 3.1. *The treewidth of a k-clique is TreeWidth $K_k = k - 1$.*

Proof of Lemma:

Clearly TreeWidth $K_k \leq k - 1$. Suppose that TreeWidth $K_k < k - 1$, and let H be a minimum-width covering hyperforest with tree decomposition $T(H)$. Since H has no hyperedge of size k, it does not have a single hyperedge covering all vertices in G, so there exist three vertices $u, v, w \in G$ that are not all included in a single hyperedge of H. Since all edges must be covered by H, for each pair of the three vertices, there must be a hyperedge containing both of them: $x, y \in h_1; y, z \in h_2; x, z \in h_3; h_1, h_2, h_3 \in H$. Consider the location of h_1, h_2, h_3 in the tree $T(H)$, and the three paths between them. Since T is a tree, there must be a node in T, i.e. hyperedge $h' \in H$, in which the three paths in T meet. The node h' might be one of h_1, h_2, h_3 or a different hyperedge. By the separation property of $T(H)$, $\{u\} = h_1 \cap h_2 \subset h'$ and similarly also for v and w, so $u, v, w \in h'$, contrary to the earlier argument.

□

Moreover, following this argument, any covering hyperforest H of a graph G must cover all the cliques in G.

Lemma 3.2. *If G is a subgraph of G' ($G \subset G'$), then TreeWidth $G \leq$ TreeWidth G'.*

Proof of Lemma:

Any covering hyperforest of G' also covers G.

□

Corollary 3.3. *TreeWidth $G \geq \max$ Clique $(G) - 1$*

3.2.2 Joining hyperforests

In Section 3.1 we noted that the union of two hyperforests is a hyperforest if their vertex sets are disjoint, but may not be a hyperforest otherwise. We now extend this to a more general situation:

Lemma 3.4. *Let $H_1(V_1)$ and $H_2(V_2)$ be hyperforests with respective tree structures $T_1(H_1)$ and $T_2(H_2)$. If $s = V_1 \cap V_2$ is covered by both H_1 and H_2, then $H_1 \cup H_2$ is also a hyperforest.*

Proof of Lemma:

Let $s \subset h_1 \in H_1$ and $s \subset h_2 \in H_2$ be hyperedges covering s and consider the tree structure $T_1 \cup T_2 \cup \{(h_1, h_2)\}$.

□

Since every clique in a graph must be covered, we also have:

Corollary 3.5. *If $G_1(V_1)$ and $G_2(V_2)$ both have treewidth at most k, and $V_1 \cap V_2$ is a clique in both G_1 and G_2, then $G_1 \cup G_2$ also has treewidth at most k.*

3.2.3 Maximal hyperedges in hyperforests

We now show that it is enough to concentrate on the tree structure of the maximal hyperedges in a hypergraph.

Theorem 3.6. *If a hypergraph has a tree decomposition, that might include maximal as well as covered edges, then there is also a tree decomposition over the maximal edges only.*

Proof of Theorem:

Let $H(V)$ be a hyperforest with tree structure $T(H)$. We will show that any non-maximal hyperedge of H can be removed from the tree structure, i.e. for $h' \subset h \in H$ we will show how to modify T into $T'(H')$, a tree structure over $H' = H \setminus \{h'\}$.

Examine the path in T from h' to h. All hyperedges on this path must contain $h' \cap h = h'$, and in particular this is true for the first hyperedge, h_1, on the path, which is adjacent to h' in T. We will remove h' from T by diverting all arcs incident on h' in T, to go to h_1 instead in T'. T' remains a tree, and paths that before contained h' now contain h_1 instead. But since $h' \subset h_1$, the decomposition property still holds.

□

A tree structure T over the maximal edges of a hypergraph can always be extended to all the subset edges, by connecting in T each subset hyperedge to a hyperedge containing it. For this reason, we freely refer to subsets of edges as being part of the hyperforest, and are concerned only with the tree structure over the maximal edges.

This theorem allows us to discuss hyperforests as if they consisted only of maximal hyperedges, but freely add their covered hyperedges to the tree decomposition where necessary.

3.2.4 Minors

We have already seen, in Lemma 3.2, that taking a subgraph of a graph can only reduce the treewidth. Taking a *minor* of a graph is a more general operation than taking a subgraph, permitting investigation of more global structures in the graph. In addition to removing vertices and edges, edge *contractions* are also allowed when taking a graph minor. Whereas subgraphs can ignore the global structure and concentrate on the local structure in some part of the graph, contractions can ignore some of the local structure and concentrate on higher-level structure.

Definition 3.7 (Graph Minor). *A graph $G'(V')$ is a* minor *of a graph G if G' can be obtained from G by a sequence of any number of operations from the following:*

- *Edge removal.*
- *Vertex removal (together with all incident edges).*
- *Edge contraction: an edge (v_1, v_2) can be* contracted *by replacing both vertices with a new combined vertex that is incident on any edge on which either v_1 or v_2 were incident.*

We will extend this standard definition also for hypergraph minors. In this case, contraction is allowed for any edge covered by the hypergraph. That is, we do not need to contract all vertices of a hyperedge, and are allowed to contract only some of them. Note that any contraction of multiple vertices can be seen as multiple contractions of pairs of vertices, or 2-edges.

Lemma 3.7. *A hyperedge-contracted hyperforest is a hyperforest*

Proof of Lemma:

It is enough to study contraction of a 2-edge. Let $H(V)$ be a hyperforest and $H'(V')$ be the result of contracting the vertices v_1, v_2 into v_{12}. The new combined vertex v_{12} replaces either of the contracted vertices in any hyperedge in which either one appears.

We show that the tree decomposition on H is still valid for H'. The decomposition property holds for all vertices other than v_{12}. We need to show that it holds also for v_{12}. Let

$v_{12} \in h'_1, h'_2$ which correspond to h_1, h_2 in H. If both h_1 and h_2 contained the same vertex v_1 or v_2 in H, then the path between them also contained this vertex, and now contains v_{12}. Otherwise, without loss of generality, $v_1 \in h_1$ and $v_2 \in h_2$. Let h_12 be the hyperedge covering (v_1, v_2) in H, and consider the possibly backtracking "path"[2] from h_1 through h_12 to h_2 in the tree structure. All hyperedges along this path must contain either v_1 or v_2, and so in H' they will contain v_{12}. Since this path covers the path from h'_1 to h'_2, we have shown the decomposition property with respect to v_{12}.

Note that some hyperedges might become identical, requiring modification to the tree structure (as it is now over fewer nodes). If $h_1, h_2 \in H$ become identical, then their intersection $h_1 \cap h_2$, and thus also all hyperedges in the path between them, must contain $h_1 \cup h_2 - v_1 - v_2$. Following the argument above, the path must also include v_{12}, making all hyperedges on the path between h_1 and h_2 identical, and allowing us to collapse the path.

□

The tree structure of the minor of the hyperforest is therefore the minor of the hyperforest's tree structure formed by contracting hyperedges that differ only in which of the contracted vertices they include.

Edge removals may turn a hyperforest into a non-hyperforest, but if a hypergraph has a covering hyperforest, then the same hyperforest will still cover it after any edge or vertex removals. Combined with Lemma 3.7, we can conclude that:

Theorem 3.8. *A minor of a graph of treewidth k has treewidth at most k, i.e. the class of graphs of width at most k is closed under taking minors.*

This property is especially useful for proving high treewidth by proving the existence of a high treewidth minor, such as a clique.

3.3 Equivalent characterizations of hyperforests

In this thesis, we have chosen to define treewidth and tree decompositions through the formalization of tree structures of hyperforests. However, this is only one of many equivalent views (and definitions) of treewidth and the decomposition concept that appeared in previous work. In this section we present other characterizations and prove their equivalence.

[2]Since we allow backtracking, this is not formally a path, as defined earlier

Presenting these equivalent characterizations serves several purposes: First, we will use some of the alternate characterizations, and properties derived from them, in later sections of the thesis (See below for details). Second, we hope that through these alternate views, the reader will become more comfortable with the notions involved, and will get a better intuition for these concepts. Each reader might find a different characterization appealing to his intuition or background. Last but not least, through the presentation of these various characterizations, we hope to demonstrate the tight connection of hypertrees to many graph-theoretic concepts.

Characterizations on which later chapters depend Graham reductions, defined in Section 3.3.1, are used extensively in the proofs in Chapter 5. This is essentially the only characterization used in later sections. Triangulations (Section 3.3.4) were first discussed in Chapter 2, since they are the most common characterization used in work on Markov networks. They are referred to in later sections, but only to make connections with the presentation in Chapter 2 and the machine learning literature.

3.3.1 Graham reduceability

A 1-forest always has a vertex of degree one, sometimes called a *leaf*[3]. Removing a leaf from the 1-forest yields a new 1-forest, which again has a leaf (i.e. a different vertex of degree one). The 1-forest can thus be reduced to an empty graph by iteratively removing vertices of degree one. In fact, a 1-forest can be characterized as a graph that can be reduced to an empty graph in such a way.

A very similar characterization applies for hyperforests:

Definition 3.8. *A hypergraph $H(V)$ is* Graham reduceable *iff it is empty, or if it has some vertex $v \in V$ such that v is incident on only one maximal hyperedge of H, and the induced sub-hypergraph on $V \setminus \{v\}$ is Graham reduceable. The vertex v is called a* leaf *of H and the maximal hyperedge containing it is called v's* twig.

[3]In fact, it always has at least two such vertices.

Note that a twig of a leaf is a maximal hyperedge at the time of removal, but it might be a non-maximal hyperedge in the original hypergraph. Twigs will be discussed in greater detail in Section 3.3.2.

Theorem 3.9. *A hypergraph is acyclic iff it is Graham reduceable*

Proof of Theorem:

Every hyperforest is Graham reduceable It is enough to show that every hyperforest has a vertex that is incident on only one maximal hyperedge. Since every induced sub-hypergraph of a hyperforest is also a hyperforest, removing this vertex yields a hyperforest, and the argument can be applied iteratively.

For a hyperforest $H(V)$ with tree structure $T(H)$ over its maximal edges, let h_1 be a leaf (i.e. node of degree one) of T, and let h_2 be the only node adjacent to h_1 in T. Any vertex not unique to h_1 must also be included in h_2. Thus since h_1 is a maximal hyperedge, it cannot be contained in h_2, and therefore must have some unique vertex.

Every Graham reduceable hypergraph is acyclic Similarly, it is enough to show that if $H'(V \cup \{v\})$ is reduceable to $H(V)$ by removing v, and H is a hyperforest, then H' is also a hyperforest. Let $h' = h \cup \{v\} \in H'$ be the unique maximal hyperedge covering v in H'. The hypergraph H' is the union of H and $\{h'\}$, which we will view as a one-hyperedge hyperforest. The intersection of the vertex-sets of these two hyperforests is covered by h' in $\{h'\}$ and by h in H, so following Lemma 3.4 the union hypergraph H is acyclic.

$\square$

3.3.2 Twig sequences

An approach similar to Graham reductions is that of *twig sequences*. Twig sequences represent a reduction ordering of hyperedges, rather than of vertices as in Graham reductions.

Twigs in 1-trees are "outer" edges, connecting the tree to leaves. If done at the proper order, from the outside inwards, all edges of a 1-tree can be removed such that every edge is a twig at the time of removal. In a hypergraph a bit more care must be taken in defining such "outer" hyperedges:

Definition 3.9 (Twigs and Twig Sequences). *A hyperedge h of hypergraph H is a* twig *iff there exists another hyperedge $h' \in H$ that contains the intersection of h and the rest of H: $h \cap (\cup(H - h)) \subset h'$. The hyperedge h' is called* a branch to h.

A series of hyperedges $(h_1, h_2, \ldots, h_m)$ *is a* twig sequence *iff every hyperedge* h_i *is a twig in the hypergraph* $\{h_1, h_2, \ldots, h_i\}$*, i.e.:* $\forall_i \exists_{j<i} \forall_{k<i} h_i \cap h_k \subset h_j$

Twig sequences are presented as *sequences* and not as recursive reductions, like Graham reductions. However, these two methods of presentation (reduction sequences and recursive reductions) are essentially the same, and do not represent a real difference between Graham reductions and twig sequences. As discussed above, the real distinction is that twigs are hyperedges, whereas Graham reductions refer to vertices. The choice of method of presentation (recursive versus sequence) is based on the common presentation in the literature.

Theorem 3.10. *A hypergraph* H *is a hyperforest iff there is an ordering of its hyperedges that forms a twig sequence.*

Proof of Theorem:

Every hyperforest has a twig sequence We will show that every hyperforest has a twig, and that removing this twig yields a new hyperforest. For a hyperforest H with tree structure $T(H)$, let h be a leaf of T (a 1-tree always has leaves). We claim that h is a twig of H: as a leaf, h has only one neighbor h' in T. The path in T between h and any other hyperedge must pass through h', so h' must contain the intersection of h and any other hyperedge of H. Furthermore, note that since h is a leaf of T, it can be removed from T leaving a new tree, which is a tree structure for $H \setminus \{h\}$. Thus, removing h yields a hyperforest.

Every twig sequence is a hyperforest We will show how a twig sequence can be Graham reduced. Let h_i be a twig of $H_i = \{h_1, \ldots, h_i\}$ with branch h_j. Every vertex of h is either unique to H_i (incident on no other hyperedge in H_i) or included in h_j. The unique vertices can be Graham-reduced, leaving a hyperedge that is completely contained in h_j, and thus is not maximal and can be ignored. The resulting hypergraph is $\{h_1, \ldots, h_{i-1}\}$, which is a twig sequence. We can proceed recursively.

□

Note that the branches of a twig sequence correspond to the edges in the tree structure of the hyperforest.

A useful feature of twig sequences is that any prefix of a twig sequence represents all the connections between vertices included in it:

Lemma 3.11. *Let $H = (h_1, \dots, h_m)$ be a twig sequence and $H_i = (h_1, \dots, h_i)$ be a prefix of H. Then any hyperedge $h' \subset \cup H_i$ that is covered by H is also covered by H_i.*

Proof of lemma:

Consider a hyperedge $h' \subset \cup H_i$ such that $h' \subset h_j \in H$, $i < j$. We will show that h' is covered by H_r for every $i \leq r \leq j$, by induction on r from j down to i. Assuming h' is covered by H_r: if it is covered by h_s, $s < r$, then it is also covered by h_s in H_{r-1}. If it is covered by h_r, then let h_s be the branch to the twig h_r in H_r. Since $h' \subset h_r \cap \cup H_i$, according to the definition of a twig, h' must be covered by h_s, which is part of H_{r-1}.

$\square$

Thus, every hyperedge in the twig sequences introduces *all* the connections between the new vertices in it and the vertices already covered by the previous hyperedges in the sequence.

3.3.3 Recursive separators

Hyperforests and tree decompositions relate directly to separators in graphs. First recall the definition of a separator:

Definition 3.10 (Graph separator). *Let $G(V)$ be a graph. A vertex set $S \subset V$* separates *between vertices $u, v \in G \setminus S$ iff every path between v and u in G passes through S.*

A vertex set $S \subset V$ separates *two disjoint vertex sets $U_1, U_2 \subset V \setminus S$ iff S separates every $u_1 \in U_1$ and $u_2 \in U_2$.*

If S is a separator in $G(V)$, then the induced graph $G|_{V \setminus S}$ is not connected, and thus there is a partitioning of V into $V = S \cup U_1 \cup U_2$ such that S separates U_1 and U_2. We say that S separates $G(V)$ *into* U_1, U_2 (this is not necessarily a unique partitioning).

In a standard tree, every vertex separates the tree into the subtrees radiating out of the vertex. We will see how this can be generalized to hyperforests.

First note that the removal of an arc from a tree T separates T (not exactly in the same sense) into two subtrees, are on each side of the arc. In a graph G with a covering hyperforest H and tree decomposition $T(H)$, every arc in the tree decomposition corresponds to a separator in G: let $(h_1, h_2) \in T$ separate T into H_1, H_2. Then the intersection $s = h_1 \cap h_2$ separates T into $(\cup H_1) \setminus s$ and $(\cup H_2) \setminus s$.

Note that the separator is *not* the hyperedge, but rather the intersection of two hyperedges, corresponding to an arc in the tree decomposition. Being intersections of two non-identical hyperedges of size at most $k+1$, the separators are thus of size at most k. Of course, since the hyperedges include the intersection, the hyperedges themselves are also separators.

Consider the two induced subgraphs of G over $V_1 = \cup H_1$ and over $V_2 = \cup H_2$. These two induced subgraphs are covered by H_1 and H_2, which are both k-hyperforests. Thus, the separation can be continued recursively, separating G further and further, up to subgraphs of $k+1$ vertices or less. A covering k-hyperforest thus corresponds to a recursive separation of $G(V)$, with separators of size at most k. Note that, at each recursive stage, the separator is part of both of the resulting subgraphs.

The converse is also true: given such a recursive separation using separators of size at most k, a covering k-hyperforest can be constructed. We will formalize this in the following definition and theorem:

Definition 3.11 (Recursive Separability). *A graph $G(V)$ is* recursively separable *with separators of size at most k iff either: (1) $|V| \leq k+1$ or (2) there exists a separator $|S| \leq k$ that separates $G(V)$ into V_1, V_2 and the induced subgraphs of G on $V_1 \cup S$ and on $V_2 \cup S$ are both recursively separable.*

Theorem 3.12. *A graph G has a covering k-hyperforest iff it is recursively separable with separators of size at most k.*

Corollary 3.13. *The treewidth of a graph is the minimum k for which it is recursively separable with separators of size at most k.*

This provides an alternate characterization of treewidth, which might demonstrate the decomposability of the graph more vividly. The graph can be decomposed into small components, the “glue” between the components being small in each level of the decomposition.

Connectivity

The recursive separability shows that no part of the graph is very highly connected. This property is much stronger than the connectivity of a graph. A graph is said to be k-connected if the minimal separator is of size k. Clearly, the treewidth is at least equal to the connectivity of the graph. However, a graph with low connectivity may have a small separator separating two large cliques. Low treewidth, through the recursive separation property, guarantees a uniformly low degree of connectivity, throughout the graph. In fact:

Theorem 3.14. *A graph has treewidth at most k iff every induced subgraph of it is at most k-connected.*

Proof. If G has a covering k-hyperforest H, every induced graph of G is covered by the corresponding induced sub-hyperforest of H, guaranteeing a separator of size at most k. If every induced subgraph of G is at most k-connected, then G is recursively separable, since after each separation we can use the separator over the new subgraphs to continue the recursion. □

3.3.4 Triangulations

Perhaps the most common essentially equivalent form of hyperforests found in the literature is triangulated graphs.

Definition 3.12 (Triangulated Graph). *A graph $G(V)$ is* triangulated *iff it has no minimal cycles of more then three vertices.*

In a triangulated graph, every cycle of four or more vertices must have a *chord*, i.e. an edge connecting two non-consecutive vertices of the cycle. Because of this, triangulated graphs are sometimes referred to also as *chordal graphs*.

Definition 3.13 (Triangulation). *A* triangulation *of a graph $G(V)$ is a triangulated supergraph $G'(V) \supset G(V)$.*

Hyperforests and tree decompositions are tightly associated with triangulated graphs through a structure called a *junction tree*.

Tree decomposition of a triangulated graph

Theorem 3.15 (Junction Tree Theorem). *For any triangulated graph $G(V)$, let $\mathcal{C}$ be the set of maximal cliques in $G(V)$. Then $\mathcal{C}$ is a hyperforest, i.e. it has a tree decomposition $J(\mathcal{C})$.*

The tree decomposition over the maximal cliques, $J(\mathcal{C})$ is called the *junction tree* of the triangulated graph G.

The Junction Tree Theorem shows that the the width of $\mathcal{C}$, i.e. $\max \mathit{Clique}\,(G) - 1$ is an upper bound on the treewidth of G. But this upper bound matches the lower bound obtained in Corollary 3.3, showing that:

Corollary 3.16. *The treewidth of a triangulated graph G is $\max \mathit{Clique}\,(()G) - 1$.*

Moreover, since any covering hyperforest H of G must cover all of G's cliques, it also covers $\mathcal{C}$. The hypergraph $\mathcal{C}$ is thus the unique minimal covering hyperforest of the triangulated graph G. Note that non-triangulated graphs do not necessarily have a unique minimal covering hyperforest— different triangulations of G correspond to different covering hyperforests.

The junction tree J itself is *not* unique—there may be several tree structures over $\mathcal{C}$ for which the decomposition property holds.

Hyperforests as triangulated graphs

We showed that a triangulated graph has a natural hyperforest associated with it. This correspondence is bi-directional. A hyperforest H can be seen as a triangulated graph:

Definition 3.14 (Infrastructure graph of a hypergraph). *The* infrastructure *graph of a hypergraph H is the graph G that contains all the possible 2-edges covered by H: $G = \{(u,v)|\exists_{h\in H} u,v \in h\}$*

In the infrastructure graph, every hyperedge becomes a clique.

Theorem 3.17. *The infrastructure graph of a hyperforest is a triangulated graph.*

Proof of Theorem:

Let H be a hyperforest with tree decomposition T, and let G be its infrastructure graph. Assume G is not triangulated, and let $c = (v_1, v_2, \ldots, v_k, v, 1)$ be a chordless cycle in G.

Every edge in the cycle must be covered by a hyperedge in H, but no non-consecutive vertices can lie in the same hyperedge (otherwise the edge between them is a chord). This implies there are k distinct hyperedges $h_1, \ldots, h_k \in H$, each containing the corresponding edge[4] $(v_i, v_{i+1}) \in h_i$, but not other vertices from c. Note that there may be several hyperedges that cover each edge in c, and we choose only one covering hyperedge for each edge.

Consider the hyperedges $h_1, \ldots, h_k$ as nodes in T and the paths in T between them. Let p_i be the path between h_{i-1} and h_i in T. From the decomposition property, all hyperedges in the path p_i must contain $h_{i-1} \cap h_i$, which includes v_i, so no two non-consecutive paths can intersect (or else the hyperedges in their intersection contain non-consecutive vertices of c, forming a chord).

We now argue that by choosing the covering hyperedges h_i appropriately, we can assume without loss of generality that consecutive paths p_i, p_{i+1} do not intersect (except at the endpoint h_i). If the paths do intersect, all hyperedges in their intersection $p_i \cap p_{i+1}$ must include both v_i (because they are in p_i) and v_{i+1} (because they are in p_{i+1}). Thus, every hyperedge in $p_i \cap p_{i+1}$ covers the edge (v_i, v_{i+1}) and is a candidate for h_i. To eliminate any intersection, we can choose the hyperedge in the intersection that appears first in p_i, as our "new" h_i, truncating p_i and p_{i+1} appropriately.

Therefore, $p_1, p_2, \ldots, p_k, p_1$ are k-connected, but not intersecting, paths in T, forming a cycle $h_k \xrightarrow{p_1} h_1 \xrightarrow{p_2} h_2 \cdots h_{k-1} \xrightarrow{p_k} h_k$ in T, contrary to T being a tree.

□

We see that the correspondence between triangulated graphs and hyperforests is very tight. In fact, there is a one-to-one correspondence between triangulated graphs and hyperforests in which all hyperedges are maximal (i.e., in which no hyperedge is covered by

[4]Here, and later in the proof, we omit the implied modulo on the index

another hyperedge). In such cases Theorems 3.15 and 3.17 give the two directions of the one-to-one correspondence.

Moreover, a graph G is covered by a hyperforest H if and only if G is a subgraph of the infrastructure graph of H. Combining this observation with the above paragraph, we see that a graph G is covered by a hyperforest H if and only if the infrastructure graph of H is a triangulation of G. This leads us to the following theorem:

Theorem 3.18. *For a graph G, the treewidth of G is equal to the minimum over all triangulations G' of G, of the maximal clique size in G', minus one:*

$$width(G) = \min_{\text{triang } G' \supseteq G} \max \textit{Clique}\,(()G') - 1 \tag{3.1}$$

This theorem provides an alternate definition of treewidth that does not use the notions of a hypergraph or tree decompositions.

3.4 Further Reading

We point the interested reader to further topics related to concepts discussed in this section. These topics are beyond the scope of this thesis and are not used in the following chapters.

3.4.1 Algorithms for calculating treewidth and finding tree decompositions and triangulations

Calculating the treewidth of a graph is NP-hard in general [CP87]. However, for a constant k, deciding if a graph is width k, and even finding a covering k-hyperforest, can be done in linear time [Bod96]. However, the dependence on k is so extreme that the linear time algorithms are impractical even for very small k (as low as 4 or 5). Several approximation algorithms [BGHK95] and heuristics [SG97] have been suggested for finding narrow covering hyperforests (or equivalently, triangulations) with better dependence on k. See [Bod97] for further references.

3.4.2 Graph minors

Robertson and Seymour show that classes of graphs which are closed under taking minors can be characterized by “forbidden minors”:

Theorem 3.19 (Wagner’s “conjecture”). *If a class $\mathcal{G}$ of graphs is closed under taking minors, there exists a finite set of graphs $\mathcal{F}$, called the* obstruction *of $\mathcal{G}$, such that for every graph G, $G \in \mathcal{G}$ iff there is no graph in $\mathcal{F}$ that is a minor of G.*[5]

Since graphs of bounded treewidth are closed under taking minors, we can characterize them, for any bound k, by their obstruction. This characterization is sometimes useful. For example, the first quadratic time algorithm[6] for deciding if a graph has tree width at most k, for fixed k, was based on the obstruction characterization [RS95].

3.4.3 Applications

Tree decompositions are useful in many applications in which it is beneficial to decompose the graph into simple elements on which computation can be done independently, propagating information along the tree structure. The main application of interest in this thesis is Markov networks, which were introduced in Chapter 2. The connection between Markov networks and tree decompositions will be discussed in detail in Section 5.1.

In this section we mention several other applications which may be of interest to the reader. For more applications see [Bod93].

Sparse matrix inversion

If the rows and columns of a sparse matrix can be partitioned such that each group of rows has non-zero entries only in one group of columns, and vice versa, the matrix can be inverted by inverting each such block separately. If the non-zero blocks overlap, this can still be done as long as the blocks form a hyperforest. Note that in this application, the

[5]More formally, no graph of $\mathcal{F}$ is allowed to be isomorphic to a minor of G

[6]And in fact, first poly-time algorithm where the exponent does not depend on k

dependence on the clique sizes is cubic, or even slightly sub-cubic, and not exponential as in most other application.

Combinatorial queries on graphs

Many combinatorial problems on graphs, which in general are hard (NP-hard and even PSPACE-hard) can be decided in polynomial, and sometimes even linear, time on graphs of bounded treewidth, using the covering hyperforest's tree decomposition [AP89]. More generally, any predicate that can be expressed in (generalized) monadic second order logic over the graph can be solved in linear time for graphs of bounded tree width [Cou90]. The dependence on the treewidth is, of course, exponential.

Chapter 4

Hypertrees and the Maximum Hypertree Problem

We are now ready to state the central combinatorial optimization problem this work is concerned with— the maximum hypertree problem. In this chapter we present the problem (in Section 4.1) and its variants and prove that, even in a very restricted form, it is NP-hard (4.2).

In Section 4.3 we present some properties of hypertrees that might be used in approaching the maximum hypertree problem, but that are not used in our algorithms. This last section can be skipped without loss of understanding of the rest of the thesis.

4.1 Problem Statement: The Maximum Hypertree Problem

As we will see, we would like to find a maximum weight hypertree. When working with standard graphs, a weight function assigns a weight to each potential edge, i.e. pair of vertices, and the weight of the graph is the sum of the weights of its edges. However, for our applications, as will be described in Section 5.3, it is essential to assign weights also to larger subsets of edges. A hyper-weight function assigns a weight to subsets of vertices of arbitrary size, and the weight of a hypergraph is the sum of *all* the weights of edges *covered* by it: $w(H) = \sum_{h \in H} w(h)$.

The maximum hypertree problem is:

Given as inputs:

- An integer treewidth k.
- A vertex set V and a weight function $w : \binom{V}{k+1} \to \Re$ on hyperedges of size up to and including $k+1$.

Find a hyperforest $H(V)$ of width at most k that maximizes $w(H) = \sum_{h \in H} w(h)$.

For a non-negative weight function, a k-hyperforest can always be expanded to a k-hypertree with at least as much weight. And so a maximum weight hyperforest can always be taken to be a hypertree. If some candidate edges have negative weight, this might not be the case.

However, we limit our attention only to weight functions which are monotone on k-hyperforests, i.e. such that for any k-hyperforest H and any sub-hyperforest of H' of H, $w(H) \geq w(H')$. The maximum weight hyperforest will thus be also maximal with respect to covering, and so will be a k-hypertree. It is enough to limit our attention to hypertrees, and we refer to this problem as the maximum hypertree problem and not the maximum hyperforest problem.

In practice, in the algorithms presented here, we will use only a weaker property, requiring monotonicity only on cliques of at most $k+1$ vertices. Since such cliques are hyperforests, monotonicity on cliques follows from monotonicity on hyperforests.

When $k = 1$, the maximum hypertree problem is simply the maximum spanning tree problem, which is equivalent to the minimum spanning tree problem, and can be solved in polynomial time [CLR89].

Since a weight can be specified for each possible hyperedge of size up to $k+1$, the input can be of size $\Theta(n^{k+1})$, meaning any algorithm will, at best, have an exponential dependence on k. As most uses of tree decompositions are exponentially dependent on k, this is not an overly exaggerated input size.

The weight function assigns a weight to *every* subset of at most $k+1$ vertices. Every such vertex set will be referred to as a *candidate edge*.

We will also discuss variants in which there are only zero/one weights, and in which the weights are only on edges of a specific size.

4.2 Hardness of Maximum Hypertree

We show that the Maximum Hypertree problem with nonnegative weights (and so also with monotone weights) is NP-hard, even for a constant $k = 2$ (for $k = 1$ this is the maximum spanning tree problem, which can be solved in polynomial time). Furthermore, it is hard even when the weights are only on 2-edges, and the weights are only zero or one. Under this restrictions, the problem can also be formulated as: given a graph G, find a subgraph $G' \subset G$ of treewidth at most 2, maximizing the number of edges in G'.

We first relax the zero/one weight restriction, and show a reduction from 3SAT to the 2-maximum hypertree problem, with integer weights on 2-edges.

Theorem 4.1. *The maximum hypertree problem is NP-hard, even for treewidth two, and weights only on 2-edges.*

To prove the hardness, we show a reduction from 3SAT.

Overview of the (integer-weight) reduction

Given a 3CNF formula ϕ over n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$ we will construct a vertex set V and a weight function w over V, such that ϕ is satisfiable iff there exists a 2-hypertree over V with weight above some specified threshold.

The construction will consist of three layers:

- A core structure with high weight that must be included in any hyperforest that passes the threshold. The core structure by itself has treewidth two.

- Two “assignment” edges for each variable, corresponding to the two possible truth assignments. The core structure combined with both edges has treewidth three, guaranteeing that only one “assignment” edge is chosen per variable. The weights of these edges are such that at least one per variable must be included to reach the threshold.

- For each literal appearing in a clause, a “satisfaction” edge of weight one. The core construction, together with an “assignment” edge and a disagreeing “satisfaction” edge have treewidth 3, guaranteeing that only correct literals can satisfy a clause. Additionally, the core structure combined with two or more “satisfaction” edges for the same clause has treewidth three. This is needed to prevent counting the satisfaction of the same clause twice.

We will show that if a satisfying assignment exists, then a graph consisting of the core structure, the “assignment” edges corresponding to the assignment, and one correct “satisfaction” edge for each clause, has a covering 2-hyperforest, and weight exactly equal to the threshold. On the other hand, we will show that any treewidth 2 graph with weight equal to or exceeding the threshold must include the entire core structure, exactly one of each pair of “assignment” edges, and a correct “satisfaction” edge for each clause, thus demonstrating a satisfying assignment.

Details of the reduction

The vertices of V are:

- Two vertices O and A.

- Three vertices x_i, x_i^T, x_i^F for each variable.

- A vertex c_j for each clause.

The weights are as follows:

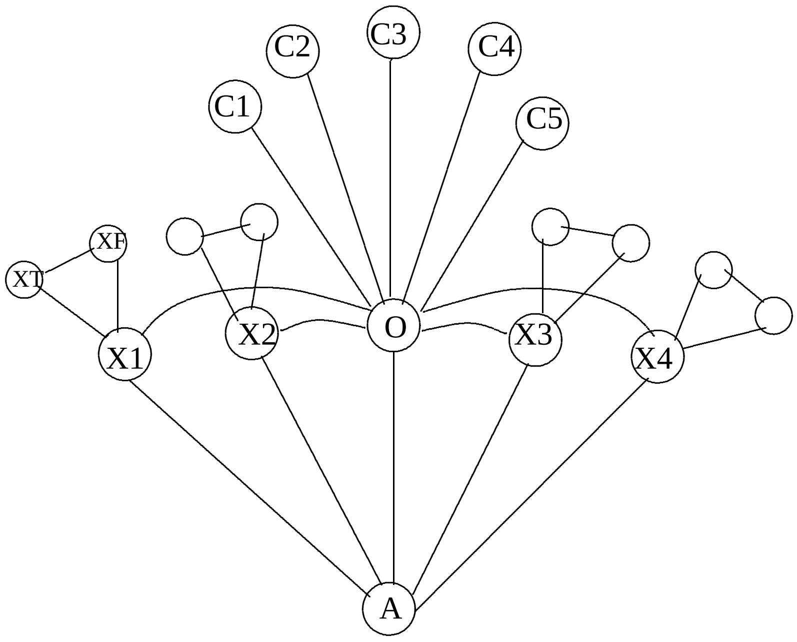

Figure 4-1: The "core" structure

- The following "core" edges will have weight $w_c = 10nm$:
  - The edge (O, A).
  - For each vertex, the edges $(x_i, x_i^T), (x_i, x_i^F), (x_i^T, x_i^F), (x_i, O), (x_i, A)$.
  - For each clause, the edge (O, c_j).

  The "core" structure is illustrated in Figure 4.2.

- For each variable, the two "assignment" edges (x_i^T, O) and (x_i^F, O) will have weight $w_a = 4m$.

- For each literal $x_i = \alpha$ in clause c_j, the "satisfaction" edge (x_i^α, c_j) will have weight one.

- All other candidate edges have weight zero.

We will require a threshold of $T = 5nw_c + w_c + nw_a + m = (5n+1)10nm + 4nm + m$. We now show that ϕ is satisfiable iff there exists a hypertree $H(V)$ such that $w(H) \geq T$.

If ϕ is satisfiable, then $\exists_{H(V)} w(H) = T$

Consider the hypergraph H which includes the following hyperedges:

- For each variable x_i the hyperedges (A, O, x_i) and (O, x_i, x_i^α), corresponding to the value α assigned to x_i in the satisfying assignment.
- Each clause c_j has at least one correct literal. Choose one such literal $x_i = \alpha$ and include the hyperedge (O, c_j, x_i^α).

This hypergraph covers all the "core" edges, one of each pair of "assignment" edges, and one "satisfaction" edge per clause, exactly attaining the required threshold. To see that H is acyclic, consider the following tree-structure T over its maximal hyperedges, and the non-maximal hyperedge (O, A):

- The neighbors of hyperedge (O, A) in T are the hyperedges (A, O, x_i), for all variables x_i.
- For each variable x_i which is assigned value α, the hyperedge (O, x_i, x_i^α) is adjacent to (A, O, x_i) in T.
- Each hyperedge of the form (O, c_j, x_i^α) in H, is adjacent to (O, x_i, x_i^α) in T.

If $\exists_{H(V)} w(H) \geq T$, then ϕ is satisfiable

First note that H must cover all "core" edges: the threshold is greater than the total weight of core edges, and the combined weight of all non-"core" edges is less then a single "core" edge, and so no "core" edge can be ignored.

Similarly, since the total weight of all "satisfaction" edges is less than a single "assignment" edge, at least n "satisfaction" edges must be covered. But if the two "assignment" edges for the same variable x_i are covered, then together with "core" edges, the four-clique $\{O, x_i, x_i^T, x_i^F\}$ is covered, contrary to H being a 2-hyperforest. Thus, exactly one "assignment" edge is covered for each variable. The covered "assignment" edges imply an assignment. We will show that this assignment satisfies ϕ.

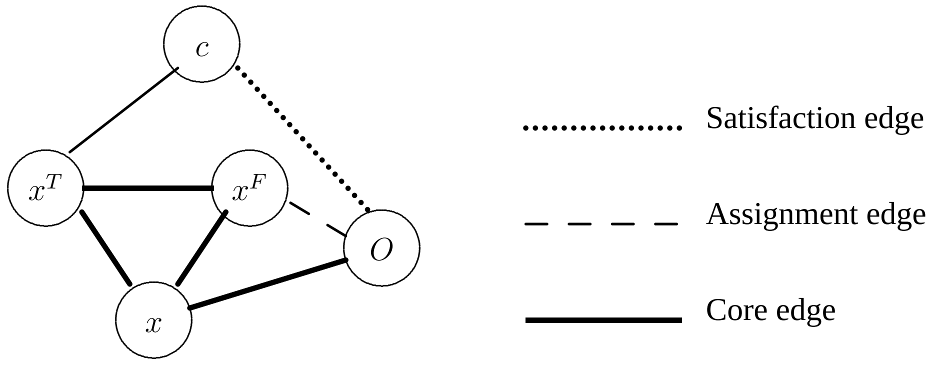

Figure 4-2: Subgraph of width three resulting from inconsistent clause satisfaction

After accounting for the "core" edges, and n "assignment" edges, m more "satisfaction" edges are required to reach the threshold. We will first see that if a satisfaction edge is covered, the corresponding clause is indeed satisfied. If a satisfaction edge (x^T, c) (w.l.o.g.) is covered, and the "wrong" assignment edge (x^F, O) is also covered, then the subgraph shown in Figure 4.2 is covered. Contracting the edge (c, O) yields a four-clique minor, contrary to H being a 2-hyperforest. Thus, (x^T, O) must be covered, x be assigned T, and the clause c satisfied.

To see that all clauses are satisfied, we will show that each of the n covered "satisfaction" edges satisfies a different clause. If two "satisfaction" edges (x_1^α, c) and (x_2^β, c) of the same clause c are covered, the subgraph shown in Figure 4.2 is covered. Contracting (c, x_2^β, x_2, A) yields a four-clique, contrary to H being a 2-hyperforest.

□

Zero/one weights

We now show how to extend the reduction to zero/one weights:

Lemma 4.2. *The 2-maximum-hypertree problem, with integer weights on 2-edges, can be reduced to the the 2-maximum-hypertree problem, with zero/one weights on 2-edges. The reduction is pseudo-polynomial, i.e. polynomial in the value of the weights.*

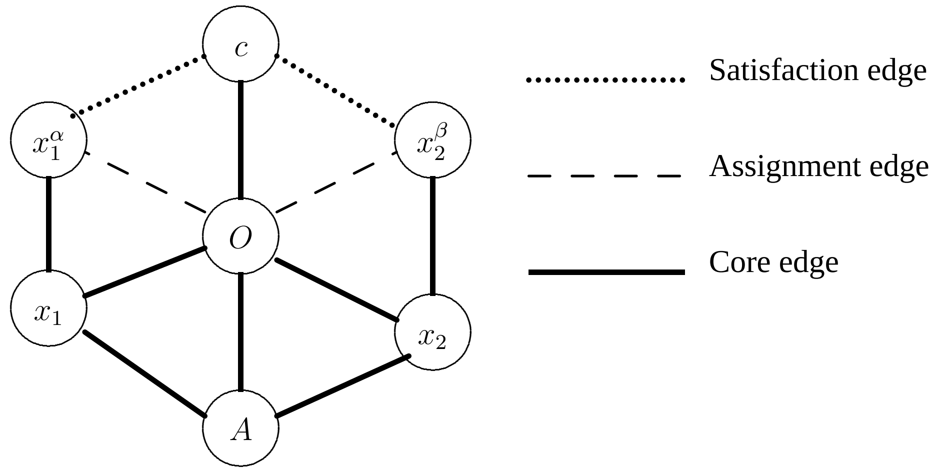

Figure 4-3: Subgraph of width three resulting from covering two "satisfaction" edges to the same clause

Proof. Given a weight function w over candidate 2-edges in vertex set V, we show how to construct a graph $G'(V')$, representing a zero/one weight function w' over V', such that the maximal hypertree with respect to w' is a constant additive distance away from the maximal hypertree with respect to w.

Overview of reduction We would like to replace each candidate edge (v_1, v_2) over V, with $w(v_1, v_2)$ parallel edges in G'. Because we are seeking a simple graph, we cannot do that directly. Instead, we will replace (v_1, v_2) with $w(v_1, v_2)$ parallel paths.

The straight-forward approach of adding adding $w(v_1, v_2)$ intermediate points, with a path of length two through each one of them, does not work. One can collect half the weight, without suffering any consequences, by covering only the edges incident on v_1, but not completing any path to v_2.

To prevent this possibility of eating the cake while leaving it whole, we make the edges from v_1 to the intermediate points mandatory, regardless of whether (v_1, v_2) is covered. The collectible weights are only between the intermediate vertices and v_2, and covering even one of them is enough to guarantee a minor in which v_1 and v_2 are adjacent.

Figure 4-4: The subgraph replacing a single edge of weight four between v_1 and v_2

We create mandatory edges between v_1 and an intermediate vertex u, by adding yet another level of intermediate points. But this time so many intermediate points, in fact more then the whole weight in w, that any maximal hypertree must cover nearly all of them (see Figure 4.2). In any case at least one path from v_1 to u must be covered, for every intermediate vertex u on every edge (v_1, v_2). The weight on these mandatory edges causes an additive constant between the weight of maximal trees with respect to w and w'.

Any 2-hyperforest H over V, can be extended to a 2-hyperforest H' over V', which covers all mandatory edges, and all additional edges corresponding to 2-edges covered by H. The weight $w'(H')$ is thus equal to $w(H)$, plus the weight of all mandatory edges, which is a predetermined constant.

We constructively show how the maximal hypertree H' with respect to w', has a minor H which covers weight of at least $w'(H')$ minus the weight of all mandatory edges. As discussed above, H' must cover at least one pair of edges $(v_1, t), (t, u)$ for every intermediate vertex u on every edge (v_1, v_2). For every v_1, v_2, u, contract those two edges, and delete any other second level intermediate vertices. The resulting minor is H. Consider a non-mandatory edge (u, v_2) covered by H'. This edge was part of a path corresponding to an edge (v_1, v_2), but v_1 and u have now been contracted together, so (v_1, v_2) is covered by H. For every (v_1, v_2), there are only $w(v_1, v_2)$ such intermediate vertices, and so the $w(H)$ contains all the non-mandatory weight from H'.

Since H is a minor of H', it has treewidth at most 2, i.e. covered by a 2-hypertree, the weight of which is at least the weight of H.

□

Corollary 4.3. *The maximum hypertree problem is NP-hard, even for $k = 2$, zero/one weights, and weights only on 2-edges.*

Proof of Corollary:

Although the reduction in Lemma 4.2 is pseudo-polynomial, note that the weights in the reduction of Theorem 4.1 are polynomial in the size of the 3CNF formula.

□

Note that both reduction presented have a significant additive constant, and are thus not L-reductions, and do not show hardness of approximation.

4.3 Properties of Hyper Trees and Maximal Hyper Trees

We present, without proof, some combinatorial properties of hypertrees. We discuss how, like 1-trees, hypertrees display a rather regular structure, with a constant number of hyperedges, and a balance between acyclicity and connectedness. This regular structure allows for equivalent alternate characterizations of hypertrees, which can be used in setting the constraints of optimization algorithms. However, we are yet unable to leverage these constraints for efficient algorithms to solve, or at least approximate, the maximum hypertree problem. We present the properties for reference only; they are not used in subsequent sections or chapters.

For a vertex set V with at most $k+1$ vertices, the hypergraph $\{V\}$ consisting of the full hyperedge (the hyperedge containing all vertices) is a hyperforest of width at most k. Since this hypergraph obviously covers any other hypergraph, it is also the only k-hypertree. In the discussion that follows, we will always assume there are at least $k+1$ vertices.

We would like to investigate what properties of 1-trees extend to hypertrees. Recall the following equivalent definitions for a 1-tree:

- A maximal acyclic graph.
- An acyclic graph with $n-1$ edges.
- A minimal connected graph.
- A connected graph with $n-1$ edges.
- An acyclic connected graph.

The definition we have used so far is a generalization of the first of these definitions. We will see that all of these equivalent characterizations can be generalized to hypertrees, if "connectedness" is generalized properly.

We first note that a hypertree is always a regular hypergraph, i.e. all maximal hyperedges in a k-hypertree are of size exactly $k+1$. Furthermore, a k-hypertree over n vertices has *exactly* $n-k$ regular (i.e. size $k+1$) edges. We get the second equivalent definition:

- a hypergraph with maximum edge-size $k+1$ is a k-hypertree if it is acyclic and has exactly $n-k$ edges of size $k+1$.

It is also useful that the number of covered hyperedges of each size is constant. The number of edges of size $r \leq k$ covered by a k-hypertree is exactly $\binom{k}{r}\left(n-k-1+\frac{k+1}{r}\right)$.

In order to generalize connectivity, we introduce the following definitions:

Definition 4.1. *Two hyperedges of a hypergraph $h_1, h_2 \in H$, both of size $k+1$, are said to be* strongly adjacent *iff the size of their intersection is k: $|h_1 \cap h_2| = k$. A series* *of hyperedges* $(h_1, h_2, \ldots, h_r)$ *is said to be a* strongly connected edge path *in a regular hypergraph* H *iff consecutive hyperedges are strongly adjacent. A regular hypergraph* $H(V)$ *is said to be* strongly connected *iff for every two vertices* $u, v \in V$*, there exists a strongly connected edge path in* H*, such that* u *appears in the first hyperedge of the path, and* v *appears in the last.*

Hypertrees are always strongly connected, and the following hold:

- A hypergraph is a k-hypertree iff it is minimal strongly-connected $k+1$-regular hypergraph.
- A hypergraph is a k-hypertree iff it is a strongly-connected $k+1$ regular hypergraph with exactly $n-k$ regular (size $k+1$) edges.

Chapter 5

Equivalence of Markov Network Projections and Maximum Hypertrees

In this chapter we describe how the Markov network projection problem can be reduced to a combinatorial problem of finding the maximum weight hypertree. That is, how the learning problem can be solved using the combinatorial problem. We also describe the reverse reduction and its implications about the hardness of the learning problem, and discuss the connections between approximate solutions of the two problems.

In Section 5.1 we set the ground for the reductions by establishing the connection between the decomposition of Markov networks and the covering hyperforest of their underlying graph. This connection was first alluded to in Section 2.2.3, where triangulated Markov networks were discussed. In particular we prove the factoring specified in equation (2.6). Section 5.1 is based on previously known results [WL83], although the presentation here is different in that it uses the notion of hyperforests to discuss the cliques of a triangulated graph.

The rest of the chapter presents original results, relating the Markov network projection problem to the maximum hypertree problem.

The connection between hypertrees and Markov network projections was previously addressed by Malvistutu [Mal91]. However, hyperedge weights were not discussed in that work, and to the best of our knowledge this is the first formulization of the maximum hypertree problem in terms of sum-of-hyperedge-weights. This formalization leads to a purely combinatorial problem (finding maximum hypertrees) which, as we show here, is equivalent to the Markov network projection problem. To the best of our knowledge, this is the first demonstration of a combinatorial problem proven to be equivalent to the Markov network projection problem. This equivalence facilitates analysis of the hardness and of the approximability of finding Markov network projections.

5.1 Decomposition Over Hyperforests

In this section we show how a Markov network is factorizable over a covering hyperforest of its dependency graph. As we will define shortly, a distribution is said to be *factorizable over a hypergraph* if it can be written as a product of *factors* corresponding to the hyperedges of the hypergraph. For Markov networks, we present an explicit factorization and prove it. We also discuss the converse, i.e. whether a factorizable distribution is a Markov network.

The factorization presented here is essentially the same as the factorization over cliques for a triangulated graph, given but not proved in equation (2.6) of Section 2.2.3. In fact, since the cliques of a triangulated graph form a hyperforest (Theorem 3.15), we essentially prove (2.6) and the Hammersly-Clifford Theorem (Theorem 2.1) for triangulated graphs.

Definition 5.1. *A distribution P over a random vector X_V is* factorizable over a hypergraph $H(X)$ *if the distribution can be written as:*

$$P(x) = \prod_{h \in H} \phi_h(x_h) \tag{5.1}$$

for some set $\{\phi_h\}_{h \in H}$ of factors, one factor for each hyperedge of H. A factor ϕ_h *corresponding to hyperedge h is a function only over the outcomes of the variables X_h (i.e. those indexed by vertices in h).*

Note that it is enough to specify factors only over maximal hyperedges, since sub-edges can always be “swallowed” inside them. However, in many cases, it is convenient to give a factorization in terms of all covered, not necessarily maximal, hyperedges.

We will see how Markov networks are tightly connected with factorizations over hypergraphs, and especially hyperforests.

Theorem 5.1. *Let X be a Markov network over a graph $G(X)$ with covering hyperforest $H(X)$. Then the distribution of X is factorizable over H, with the following factors:*

$$\phi_h(x_h) \doteq \frac{\Pr(x_h)}{\prod_{h' \subset h} \phi_{h'}(x_{h'})} \tag{5.2}$$

Note that a product of factors of all covered, not necessarily maximal, hyperedge should be taken. Similar to equation (2.6) in Section 2.2.3, the factors (5.2) are recursively defined, factors of larger hyperedges being based on the factors of their sub-edges.

Proof. By Theorem 3.9, H is Graham reduceable. We will prove by induction on the reduction, that the factors given by (5.2) multiply out to the correct distribution. For an empty graph, the factorization is empty, and trivially true. We will show that if H has a leaf v and the factors for the induced subgraph $H[V-v]$ given by (5.2) multiply out to the marginal distribution of X_{V-v}:

$$\Pr(x_{V-v}) = \prod_{h \in H[V-v]} \phi_h(x_h) \tag{5.3}$$

then multiplying all the factors for H yields the complete distribution.

Since X is a Markov network, X_v only depends on its neighbors in G. Since H covers G, all neighbors in G are also neighbors in H. But v is a leaf, and so there is a unique maximal hyperedge $h \in H$ that includes it, and so also all its neighbors:

$$\begin{aligned}\Pr(x) &= \Pr(x_{V-v})\Pr(x_v|x_{V-v})\\ &= \Pr(x_{V-v})\Pr(x_v|x_h)\end{aligned}$$

Using the factorization (5.3) of X_{V-v}:

$$= \prod_{h' \in H[V-v]} \phi_{h'}(x_h) \mathrm{Pr}\left(x_v | x_h\right) \tag{5.4}$$

We now have to show that the factors of the new hyperedges $H \setminus H[V-v]$ multiply out to $\mathrm{Pr}\left(x_v | x_h\right)$. The only new hyperedges are those containing v, and the only maximal hyperedge containing v is h, and so the new hyperedges are h itself, and the its sub-edges containing v:

$$\begin{aligned} \phi_h(x_h) \prod_{h' : v \in h' \subset h} \phi_{h'}(x_{h'}) &= \frac{\mathrm{Pr}\left(x_h\right)}{\prod_{h' \subset h} \phi_{h'}(x'_h)} \prod_{h' : v \in h' \subset h} \phi_{h'}(x_{h'}) \\ &= \frac{\mathrm{Pr}\left(x_h\right)}{\prod_{h' \subset h - v} \phi_{h'}(x_{h'})} \end{aligned} \tag{5.5}$$

Using the induction assumption on X_{h-v} with its induced hypergraph $H[h-v] = \{h-v\}$, we know that $\mathrm{Pr}\left(x_{h-v}\right) = \prod_{h' \subset h-v} \phi_{h'}(x_{h'})$. Inserting this in the denominator of (5.5) yields:

$$\phi_h(x_h) \prod_{h' : v \in h' \subset h} \phi_{h'}(x_{h'}) = \frac{\mathrm{Pr}\left(x_h\right)}{\mathrm{Pr}\left(x_{h-v}\right)} = \mathrm{Pr}\left(x_v | x_{h-v}\right). \tag{5.6}$$

Combining equations (5.4) and (5.6) yields the desired factorization of X_V. □

The converse

Theorem 5.1 shows that factorization over a covering hyperforest is a necessary condition for a distribution to be a Markov network. Is this also a sufficient condition ? The straightforward converse of the theorem is not always true. In the theorem statement we required that H be any covering hyperforest of G. A complete hyperforest (a hyperforest with the complete vertex set as an edge) covers any graph, and any distribution, even those incompatible with G, trivially factorizes over it.

Even if H is a minimal covering hyperforest, the converse does not necessarily hold.

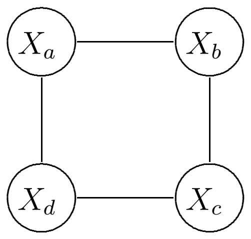

Figure 5-1: A graph on which the converse of Theorem 5.1 does not hold.

Consider the four-cycle in Figure 5.1. A possible minimal covering hyperforest is $H = \{\{a, b, c\}, \{b, c, d\}\}$. A distribution over the binary random vector (X_a, X_b, X_c, X_d) where

$$P(x_a, x_b, x_c, x_d) = \begin{cases} \frac{1}{8} & \text{If } x_b = x_c \\ 0 & \text{otherwise} \end{cases}$$

is clearly factorizable over H, but is not a Markov network over the four cycle, since X_b and X_c are dependent even given X_a and X_d. To "fix" this, the edge (b, c), which is part of the infrastructure graph of H, would need to be added to the four-cycle. The edge (b, c) is covered by H, and so distributions that include this dependency are factorizable over H.

The failure of the converse that we saw for a four cycle resulted from the covering hyperforest covering "too much", i.e. covering edges not in the original dependency graph. If, however, the graph G includes *all* edges covered by a hyperforest H, i.e. G is the infrastructure graph of H, then the converse is true and every distribution factorizable over H is a Markov network over G. [1]

As shown in Section 3.3.4, the class of graphs that are infrastructure graphs of hyperforests is exactly the class of triangulated graphs, for which we discussed factorization in Section 2.2.3. Thus, for a triangulated graph G, a distribution is a Markov network over G if and only if it is factorizable over its minimal covering hyperforest, which is its clique

[1]Note that the converse is true for G if and only if G is an infrastructure graph of H. This condition is also equivalent to H being the *unique* minimal covering hyperforest of G.

hypergraph. If it is factorizable, a possible factorization is given by equation (2.6). Note that there are many equivalent factorizations, (2.6) being only one of them.

5.2 Projection Over a Specified Graph

Even for a pre-specified graph G, projecting a distribution to a Markov network over G, is not necessarily straight-forward. If G is non-triangulated, calculating the Markov network X over G which is closest to the target distribution might require iterative calculations and cannot be specified explicitly [Pea97].

However, as indicated in Section 2.2.3, if the graph G is triangulated then the bidirectional correspondence between Markov networks over G and factorizations over its clique-forest can be used to specify the projection explicitly. We will show that the projection of a distribution $P^{\mathbf{T}}$ onto distributions factorizable over a specified hypergraph H is the unique distribution that agrees with $P^{\mathbf{T}}$ over all marginals corresponding to hyperedges in H. Because of the bidirectional correspondence, Markov networks over a triangulated G form such a class (being exactly those distributions factorizable over the clique-forest of G), and so this characterization of the projected distributions applies. We will then show the factors of the this projected distribution can be explicitly calculated.

Theorem 5.2. *For a specified hypergraph H, the projection of any distribution $P^{\mathbf{T}}$ onto the class of distributions factorizable over H, is the unique distribution $\hat{P}$ that agrees with $P^{\mathbf{T}}$ on all marginals corresponding to H:*

$$\forall_{h\in H} P^{\mathbf{T}}(X_h) = \hat{P}(X_h) \tag{5.7}$$

Note that we do not require that H be a acyclic. However, this class of distributions corresponds to a Markov networks over some graph only if H is acyclic.

Proof. The projected distribution is the one minimizing the information divergence:

$$\arg \min_{P \text{ factorizable over } H} H(P^{\mathbf{T}}||P) = \arg\min_{P} \left(\mathbf{E}_{P^{\mathbf{T}}}\left[\log P^{\mathbf{T}}\right] - \mathbf{E}_{P^{\mathbf{T}}}\left[\log P\right]\right)$$

But $\mathbf{E}_{P^{\mathbf{T}}}\left[\log P^{\mathbf{T}}\right]$ is constant (does not depend on P), and so:

$$= \arg \max_{P \text{ factorizable over } H} \mathbf{E}_{P^{\mathbf{T}}}\left[\log P\right] \tag{5.8}$$

where P is limited to distributions factorizable over H, i.e. P is any distribution that can be written as:

$$P(x) = \frac{1}{z}\prod_{h} \phi_h(x_h), \tag{5.9}$$

for any non-negative factors ϕ, and an appropriate normalizing constant z. Previously, we discussed factorizations which do not require an explicit normalizing constant. This requires the factors to multiply out to a valid distribution, and so introduces constraints on the possible sets of factors. However, we are now about to maximize over the factors, and so would prefer eliminating such constraints, and instead introducing a normalization factor that depends on all the factors:

$$z = \sum_{x}\prod_{h} \phi_h(x_h). \tag{5.10}$$

To solve the maximization problem (5.8), we shall set to zero the partial derivatives of $\mathbf{E}_{P^{\mathbf{T}}}\left[\log P\right]$ with respect to $\phi_{\check{s}}(\check{x}_{\check{s}})$, the value of the factor $\phi_{\check{s}}$ on the outcome $\check{x}_{\check{s}}$, for all $\check{s} \in H$ and for all possible outcomes $\check{x}_{\check{s}}$ of the variables $X_{\check{s}}$:

$$
\begin{aligned}
0 &= \frac{\partial \mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log P(X)\right]}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})} \\
&= \frac{\partial \mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log\left(\frac{1}{z}\prod_{h\in H}\phi_h(X_h)\right)\right]}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})} \qquad \text{[using (5.9)]} \\
&= \frac{\partial\left(\sum_{h\in H}\mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log\phi_h(X_h)\right] - \mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log z\right]\right)}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})}
\end{aligned} \tag{5.11}
$$

We will take the derivatives of the two terms separately:

$$
\frac{\partial\sum_{h\in H}\mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log\phi_h(X_h)\right]}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})} = \sum_{h\in H}\mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\frac{\partial\log\phi_h(X_h)}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})}\right]
$$

For $h \neq \check{s}$, $\log\phi_h(X_h)$ does not depend on $\phi_{\check{s}}(\check{x}_{\check{s}})$ and the derivative is zero. Accordingly, the only relevant term in the sum is $h = \check{s}$:

$$
= \mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\frac{\partial\log\phi_{\check{s}}(X_{\check{s}})}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})}\right]
$$

Similarly, $\phi_{\check{s}}(X_{\check{s}})$ will only depend on $\phi_{\check{s}}(\check{x}_{\check{s}})$ if $X_{\check{s}} = \check{x}_{\check{s}}$. This happens with probability $P^{\mathbf{T}}(\check{x}_{\check{x}})$, and in all other cases the derivative is zero:

$$
\begin{aligned}
&= P^{\mathbf{T}}(\check{x}_{\check{s}})\frac{\partial\log\phi_{\check{s}}(\check{x}_{\check{s}})}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})} \\
&= P^{\mathbf{T}}(\check{x}_{\check{s}})\frac{1}{\phi_{\check{s}}(\check{x}_{\check{s}})}
\end{aligned} \tag{5.12}
$$

As for the other term of (5.11), note that z is constant with respect to X, and so $\mathbf{E}\left[\log z\right] = \log z$ and we have:

$$\begin{aligned}
\frac{\partial \mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log z\right]}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})} &= \frac{\partial \log z}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})} = \frac{1}{z}\frac{\partial z}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})} \\
&= \frac{1}{z}\frac{\partial \sum_x \prod_h \phi_h(x_h)}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})} = \frac{1}{z}\sum_x \frac{\partial \prod_h \phi_h(x_h)}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})} \\
&= \frac{1}{z}\sum_x \frac{\partial \prod_{h\neq\check{s}} \phi_h(x_h)\cdot \phi_{\check{s}}(x_{\check{s}})}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})} \\
&= \sum_x \frac{\prod_{h\neq\check{s}} \phi_h(x_h)}{z}\frac{\partial \phi_{\check{s}}(x_{\check{s}})}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})}
\end{aligned}$$

Again, $\phi_{\check{s}}(x_{\check{s}})$ depends on $\phi_{\check{s}}(\check{x}_{\check{s}})$ only if $x_{\check{s}} = \check{x}_{\check{s}}$, while in all other cases the derivative is zero:

$$\begin{aligned}
&= \sum_{x|x_{\check{s}}=\check{x}_{\check{s}}} \frac{\prod_{h\neq\check{s}} \phi_h(x_h)}{z}\frac{\partial \phi_{\check{s}}(x_{\check{s}})}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})} \\
&= \sum_{x|x_{\check{s}}=\check{x}_{\check{s}}} \frac{\prod_{h\neq\check{s}} \phi_h(x_h)}{z}\frac{\partial \phi_{\check{s}}(\check{x}_{\check{s}})}{\partial \phi_{\check{s}}(\check{x}_{\check{s}})} \\
&= \sum_{x|x_{\check{s}}=\check{x}_{\check{s}}} \frac{\prod_{h\neq\check{s}} \phi_h(x_h)}{z}\cdot 1 \\
&= \sum_{x|x_{\check{s}}=\check{x}_{\check{s}}} \frac{\prod_h \phi_h(x)(x_h)}{z}\frac{1}{\phi_{\check{s}}(\check{x}_{\check{s}})} \\
&= \frac{1}{\phi_{\check{s}}(\check{x}_{\check{s}})}\sum_{x|x_{\check{s}}=\check{x}_{\check{s}}} \frac{\prod_h \phi_h(x)(x_h)}{z}
\end{aligned}$$

But this is exactly the distribution P, as given in (5.9):

$$\begin{aligned}
&= \frac{1}{\check{s}(\check{x}_{\check{s}})}\sum_{x|x_{\check{s}}=\check{x}_{\check{s}}} P(x) \\
&= \frac{P(\check{x}_{\check{s}})}{\phi_{\check{s}}(\check{x}_{\check{s}})} \qquad (5.13)
\end{aligned}$$

Substituting (5.12) and (5.13) in (5.11):

$$0 = \frac{P^{\mathbf{T}}(\check{x}_{\check{s}})}{\phi_{\check{s}}(\check{x}_{\check{s}})} - \frac{P(\check{x}_{\check{s}})}{\phi_{\check{s}}(\check{x}_{\check{s}})}$$

$$P^{\mathbf{T}}(\check{x}_{\check{s}}) = P(\check{x}_{\check{s}}) \tag{5.14}$$

This holds for all $\check{s} \in H$ and for all $\check{x}$, meaning that $P^{\mathbf{T}}$ and the distribution maximizing (5.8), which is the projected distribution, must agree on all marginals corresponding to hyperedges in H. □

Corollary 5.3. *For a specified triangulated graph G with clique-forest H, the projection of any distribution $P^{\mathbf{T}}$ onto Markov networks over G, is given explicitly by the following factorization over H:*

$$\hat{P}(x) = \prod_{h \in H} \hat{\phi}_h(x_h) \tag{5.15}$$

where the product is over all covered *hyperedges, and the factors are given recursively by:*

$$\hat{\phi}_h(x_h) = \frac{P^{\mathbf{T}}(x_h)}{\prod_{h' \subset h} \hat{\phi}_h(x_h)} \tag{5.16}$$

Proof. The class of distributions which are Markov networks over G is exactly the class of distributions factorizable over H, and so by Theorem 5.2, the projection $\hat{P}$ onto this class agrees with $P^{\mathbf{T}}$ over all marginals corresponding to $P^{\mathbf{T}}$. By Theorem 5.1, the factorization of a distribution over H, and in particular $\hat{P}$, is given by (5.2). Note that only marginals corresponding to hyperedges in H appear in (5.2), and so we can replace them with marginals of $P^{\mathbf{T}}$, yielding (5.16). □

5.3 Reducing Markov Network Projection to Maximum Hypertrees

In the previous section we saw that the projecting a distribution onto Markov networks over a *specified* triangulated graph can be found explicitly, and in a straight-forward way. We are now concerned with the true problem of this thesis— projecting a distribution onto Markov networks over any graph of treewidth at most k. That is, finding both a graph G and a Markov network over G, that minimizes the divergence to the target distribution. Note that finding the *projected graph* G is enough, since we already know how to find the closest distribution among the Markov networks over it. For a graph G, we call the minimum information divergence to a Markov network over G, the *information divergence to the graph* G.

A possible approach to finding the projected graph might be to enumerate over all graphs of tree width at most k, calculating the information divergence to each one, using the explicit specification of projection onto the graph. However, the exponential number of possible graphs makes this approach infeasible even for a fairly small number of variables.

5.3.1 Accumulating relative entropy using weights

A possible alternative approach, that might be more efficient, is to analyze the contribution of local elements in the graph to reducing the information divergence. Adding edges to the graph increases the space of admissible distributions, and thus reduces the information divergence. We would like to decompose the reduction in the information divergence due to “local elements”, e.g. edges, or small cliques. We might then be able to find a graph which includes many such local elements with a high contribution.

Chow and Liu [CL68] analyzed the case in which the graphs are limited to be trees. They showed that the reduction in information divergence, relative to the empty graph, can be additively decomposed to edges. The contribution of each edge of the tree is the relative information between its nodes. If a weight equal to the relative information is associated with each edge, the maximum weight tree is thus the minimum information divergence tree. Such a tree can be found efficiently.

We show here that for k-bounded treewidth graphs, edges are not enough, but $k+1$-sized local elements are enough. We will specify weights on *hyperedges* instead of edges, and show that the weight of a hyperforest is exactly the reduction in information divergence of its infrastructure graph (versus the empty graph). Therefore, by maximizing the weight of a hyperforest, the projected Markov network structure can be attained.

The minimum information divergence to a triangulated graph is:

$$H\left(P^{\mathbf{T}} \| G\right) = \min_{\substack{P \text{ is a} \\ \text{Markov net over } G}} \mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log \frac{P^{\mathbf{T}}(X)}{P(X)}\right]$$

which using the explicit factorization of the projected Markov network:

$$\begin{aligned} &= \mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log \frac{P^{\mathbf{T}}(X)}{\prod_{h\in \textit{Clique}(G)} \hat{\phi}_h(X_h)}\right] \\ &= \mathbf{E}_{P^{\mathbf{T}}}\left[\log P^{\mathbf{T}}\right] - \sum_{h\in \textit{Clique}(G)} \mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log \hat{\phi}_h(X_h)\right] \end{aligned}$$

and setting the weight for each candidate clique to $w_h = \mathbf{E}_{P^{\mathbf{T}}}\left[\log \hat{\phi}_h\right]$:

$$= -H(P^{\mathbf{T}}) - \sum_{h\in \textit{Clique}(G)} w_h \tag{5.17}$$

An important point is that the weights w_h depend *only* on the target distribution $P^{\mathbf{T}}$, and not on the structure of the graph G (as long as it is triangulated). This is because on a triangulated graph the projected factors are given by (5.2), which, unrolling the recursion, depends only on marginals of $P^{\mathbf{T}}$ inside h, and nothing else. Taking the weights $w_h = \mathbf{E}_{P^{\mathbf{T}}}\left[\log \hat{\phi}_h\right]$, the minimum information divergence to *any* triangulated graph G is given by (5.17), summing over all cliques covered by G.

As discussed, a weight w_h depends only on the marginal of $P^{\mathbf{T}}$ over h, and is given by:

$$\begin{aligned} w_h &= \mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log \hat{\phi}_h(X_h)\right] = \mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log \frac{P^{\mathbf{T}}(X_h)}{\prod_{h'\subset h}\hat{\phi}_{h'}X_{h'}}\right] \\ &= \mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log P^{\mathbf{T}}(X_h)\right] - \sum_{h'\subset h}\mathbf{E}_{X\sim P^{\mathbf{T}}}\left[\log \hat{\phi}_{h'}X_{h'}\right] \\ &= -H(P^{\mathbf{T}}(X_h)) - \sum_{h'\subset h} w_{h'} \end{aligned} \tag{5.18}$$

This provides for a simple recursive specification of the weights. Unrolling this recursion, the weight of a candidate hyperedge can also be written as a sum:

$$w_h = -\sum_{h'\subseteq h}(-1)^{|h|-|h'|}H(P^{\mathbf{T}}(X_{h'})) \tag{5.19}$$

5.3.2 Negative, but monotone, weights

Note that some of the weights might be negative. For example, weights corresponding to singletons $\{v\}$ have no sub-cliques, and therefore $w_{\{v\}} = -H(X_v) < 0$. In fact, returning to the derivation of (5.17), $\sum_{h\in \mathit{Clique}(G)} w_h = \mathbf{E}_{P^{\mathbf{T}}}\left[\log \hat{P}\right] < 0$, and so the sum of the weights is always negative. However, as more edges are added to the graph, the admissible distributions are less limited and projected distribution can become more similar to the target distribution, thus increasing $\mathbf{E}_{P^{\mathbf{T}}}\left[\log \hat{P}\right]$ (i.e. pulling it closer to zero). This means that weights of edges beyond singletons should generally have a positive contribution, representing the reduction in the information divergence, or equivalently the gain in negative cross-entropy.

There might still be multiple-vertex cliques with negative weights. For example, consider a Markov chain over three variables $X_1 \to X_2 \to X_3$. The candidate hyperedge $(1,2,3)$ has negative weight, equal to minus the mutual information between X_1 and X_3.

However, it is important to note that the weight is monotone on k-hyperforests. I.e. the weight of a k-hyperforest is greater or equal to the weight of any sub-hyperforest, and so the weight of the difference between two nested hyperforests is non-negative. Let $G' \subset G$

be nested hyperforests, then:

$$\begin{aligned}
\sum_{h\in G\setminus G'} w_h &= \sum_{h\in G} w_h - \sum_{h\in G'} w_h \\
&= \sum_{h\in G} \mathbf{E}_{X_h\sim P^{\mathbf{T}}}\left[\log\hat{\phi}_h(X_h)\right] - \sum_{h\in G'} \mathbf{E}_{X_h\sim P^{\mathbf{T}}}\left[\log\hat{\phi}_h(X_h)\right] \\
&= \sum_{h\in G} \mathbf{E}_{X_h\sim \hat{P}_G}\left[\log\hat{\phi}_h(X_h)\right] - \sum_{h\in G'} \mathbf{E}_{X_h\sim \hat{P}_G}\left[\log\hat{\phi}_h(X_h)\right] \\
&= \mathbf{E}_{X\sim\hat{P}_G}\left[\log\frac{\prod_{h\in G}\log\hat{\phi}_h(X_h)}{\prod_{h\in G'}\hat{\phi}_h(X_h)}\right] \\
&= \mathbf{E}_{\hat{P}_G}\left[\log\frac{\hat{P}_G}{\hat{P}_{G'}}\right] \\
&= H\left(\hat{P}_G\|\hat{P}_{G'}\right) \doteq H_{P^{\mathbf{T}}}\left(G\|G'\right) \geq 0 \qquad (5.20)
\end{aligned}$$

where $H\left(\cdot\|\cdot\right)$ is the, always non-negative, information divergence between distributions, and we use $H_{P^{\mathbf{T}}}\left(G\|G'\right)$ to denote the information divergence between the projection of $P^{\mathbf{T}}$ on the respective graphs.

As suggested before, this monotonicity is not surprising, since by (5.17), the difference in the weights of the hyperforests represents the difference in their minimum information divergence from the target distribution. But any distribution that is a Markov network over G' is also a Markov network over G, and so the projected distribution over G must be closer to $P^{\mathbf{T}}$ than the projected distribution over G', yielding a smaller information divergence to G, and so by (5.17) requiring the weight to be higher.

Note that these arguments hold only if both G and G' are acyclic. Otherwise (5.17) does not hold, and the weight of the graph does not represent any meaningful information quantity as the product of the factors does not multiply out to a valid distribution function, let alone the projected distribution.

Negative weights may pose a problem to many algorithm, and this monotonicity helps resolve this problem. The algorithms we present do not work well with general negative weights, but we show that they work properly with weight functions which are monotone

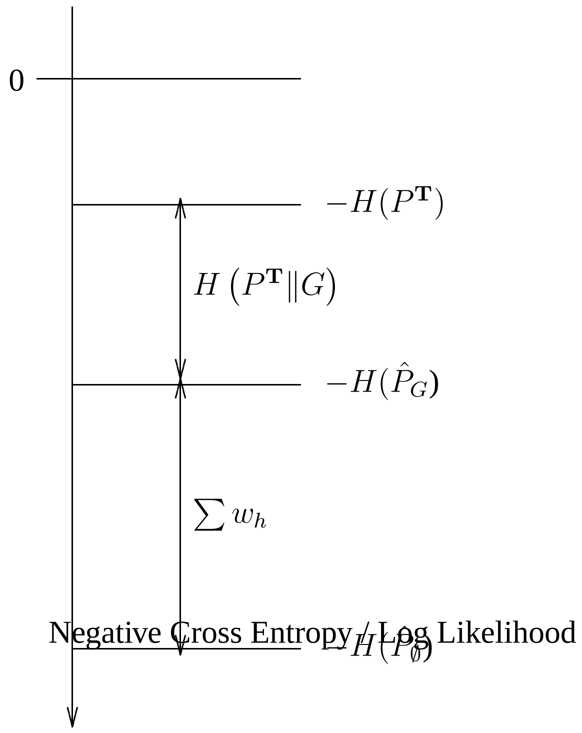
0
$-H(P^{\mathbf{T}})$
$H\left(P^{\mathbf{T}}\|G\right)$
$-H(\hat{P}_G)$
$\sum w_h$
Negative Cross Entropy / Log Likelihood
$-H(P_\emptyset)$

Figure 5-2: $H\left(P^{\mathbf{T}}\|G\right) = \left(\mathbf{E}_{P^{\mathbf{T}}}\left[\log \emptyset\right] - H(P^{\mathbf{T}})\right) - \sum_{h \in \mathit{Clique}(G), |h|>1} w_h$

on k-hyperforests.

Overall picture of the decomposition

In any graph, all single vertices will form cliques, and these would be taken in any case. The sum of these weights correspond to the negative cross entropy of the empty graph $\mathbf{E}_{P^{\mathbf{T}}}\left[\log \emptyset\right]$ and represent the base negative cross entropy, from which we only climb up. This is represented in Figure 5.3.2. We would like to minimize the information divergence, and so to maximize $\sum_{h \in \mathit{Clique}(G), |h|>1} w_h = H_{P^{\mathbf{T}}}\left(G\|\emptyset\right)$, which is the gain in negative cross entropy relative to the empty graph.

5.3.3 The reduction

We have shown how the problem of projecting a distribution onto Markov networks of bounded treewidth k (i.e. onto triangulated graphs of bounded clique size $k+1$) can be reduced to finding a maximum weight k-hyperforest for a weight function which is monotone on acyclic hypergraphs. If we knew how to find such a maximum weight hyperforest, we would choose a weight function assigning to each possible clique h, of more then a single vertex, the weight

$$w(h) = -H(P^{\mathbf{T}}(X_h)) - \sum_{h' \subset h} w_{h'} \tag{5.21}$$

defined recursively on all sets of vertices, including singletons. This weight can be calculated immediately from the empirical marginal $\hat{P}_C$. The infrastructure graph of the maximum weight hyperforest would then be the projected graph.

As discussed in Section 2.3.2, the problem of finding the maximum likelihood Markov network for a given sample can be formulated as a projection problem of the empirical distribution, and so can be reduced in the same way.

5.3.4 The complexity of the reduction

The reduction yields a maximum hyperforest problem of size $O(n^{k+1})$, as $\binom{n}{k+1}$ weights must be specified. As we have not considered the representation of the target distribution, we cannot discuss the complexity of the reduction in terms of the problem 'size', as this of course depends on the representation. We do not want to go into the issues of input representations of the distribution, except for one special case which originally motivated us: when the distribution is an empirical distribution of some sample.

The "input representation" in this case is the sample itself, of size $O(Tn\log m)$, where T is the sample size and m is the number of possible outcomes for each random variable. And so, if k is part of the input, the reduction is *not* polynomial in the sample, as it is exponential in k while the sample is independent of it. If k is constant, then the reduction is polynomial.

As the number of parameters in the resulting model, and so also the complexity of calculations on the resulting distribution, is also exponential in k, it is tempting to hope that the reduction is at least polynomial in the resulting number of parameters. This is essentially the output size of the learning problem, and practically also a bound on the input size, as one would generally not have less data then there are parameters to learn. However, this is *not* the case. The number of parameters is only $O\left(nm^{k+1}\right)$. Thus if $n >> m$, the reduction is supper-polynomial even in the resulting number of parameters.

5.4 Reducing The Maximum Hypertrees Problem to the Maximum Likelihood Markov Network Problem

It is also interesting to note that for every non-negative weight function of candidate hyperedges of a fixed size, there exists a distribution that yields weights proportional to this set of weights. I.e. the problem of finding a maximum hyperforest, at least for a non-negative weight function, can be reduced to projecting a distribution to Markov networks of bounded treewidth. Furthermore, a “small” sample can be constructed, with an empirical distribution yielding weights which are close enough to these weights, conserving the exact structure of the projected graph. I.e. the problem of finding a maximum hyperforest (for non-negative weights) can also be reduced to finding a maximum likelihood Markov network for empirical data.

This reduction is weak, in the sense that the sample size needed to produce specific weights is polynomial in the *value* of the weights (and so exponential in the size of their representation). Still, using hardness results from Chapter 5, this pseudo-polynomial reduction is enough in order to show NP-hardness of finding a maximum likelihood Markov networks of bounded treewidth, even for treewidth two.

5.4.1 A distribution yielding desired weights

For a given weight function $w : \binom{V}{k+1} \to [0,1)$ on candidate edges of size exactly $k+1$, we will consider each vertex as a binary variable, and construct a distribution P_w over these variables. The distribution will be such that using it as a target distribution in (5.21) will yield weights w' proportional to w.

We will assume, without loss of generality, that $\forall_C w(C) < 1$.

The distribution P_w will be a uniform mixture of $\binom{n}{k+1}$ distributions P_w^h, one for each $h \in \binom{V}{k+1}$. Each such P_w^h will deviate from uniformity only by a bias of $r(h)$ in the parity of the variables in h. We show below how to select $r(h)$ according to $w(h)$. Explicitly:

$$P_w^h(x) = \begin{cases} \frac{1+r(h)}{2^{|V|}} & \text{If } \sum_{v\in h} x_v \text{ is odd} \\ \frac{1-r(h)}{2^{|V|}} & \text{If } \sum_{v\in h} x_v \text{ is even} \end{cases} \tag{5.22}$$

This results in a mixed distribution P_w in which all marginals over at most k variables are uniform (and therefore have zero corresponding weight), while the marginal over a hyperedge h of size exactly $k+1$ has a bias of $b = \frac{r(h)}{\binom{n}{k+1}}$. The corresponding weight is therefore

$$\begin{aligned} w'(h) &= -H(X_h) - \sum_{h'\subset h} w(h) \\ &= -H(X_h) - \sum v \in h(-H(X_v)) \\ &= |h| \times 1 - H(X_h) \\ &= (k+1) + \sum_{x_h} P_w(x_h) \log P_w(x_h) \\ &= (k+1) + 2^k \frac{1+b}{2^{k+1}} \log \frac{1+b}{2^{k+1}} + 2^k \frac{1-b}{2^{k+1}} \log \frac{1-b}{2^{k+1}} \\ &= (k+1) + \tfrac{1}{2}\left((1+b)\log(1+b) + (1-b)\log(1-b)\right) - (k+1) \\ &= \tfrac{1}{2}\left((1+b)\log(1+b) + (1-b)\log(1-b)\right) \end{aligned} \tag{5.23}$$

Using the natural logarithm and taking the Taylor expansion:

$$
\begin{aligned}
&= \tfrac{1}{2}\left((1+b)\sum_{i=1} \tfrac{(-1)^{i-1}}{i} b^i + (1-b)\sum_{i=1} \tfrac{(-1)^{i-1}}{i}(-b)^i\right) \\
&= \sum_{i=2 \text{ even}} \frac{1}{i(i-1)} b^i \\
&= \frac{b^2}{2} + O(b^4) = \frac{1}{2\binom{n}{k+1}^2} r(h)^2 + O(r(h)^4) \qquad (5.24) \\
& \qquad (5.25)
\end{aligned}
$$

Choosing $r(h)$ to be approximately $\sqrt{w(h)}$ (or more precisely, the inverse function of (5.23)) yields weights proportional to w.

5.4.2 A sample yielding desired weights

We have shown a distribution that produces weights proportional to any desired non-negative weight function. However, since this the biases in this distribution might be irrational (and in fact if the weights are rational, the biases must be irrational, being the inverse of (5.23)), there is no finite sample that has such a distribution as its empirical distribution.

However, we will show a finite sample that results in weights which are close enough to the desired weights, such that the optimal structure is conserved. Given a rational weight function w, we will show a sample with empirical distribution $\hat{P}_w$ that produces weights $w''(h) = w'(h) + e(h)$ such that w' are proportional to w, and $\sum_h |e(h)| < \delta$ where $\delta = \min_{h_1,h_2,w'(h_1)\neq w'(h_2)} |w'(h_1) - w'(h_2)|$ is the granularity of the weights. This is enough, since the w' and w'' weights of the optimal hypergraph will be within δ, less then the possible difference due to taking edges with differing weights.

Constructing a sample with rational biases

We first show how to construct a sample which yields an empirical distribution similar in structure to P_w, with rational biases on $k+1$ candidate edges. I.e. for any mapping $h \mapsto \frac{p_h}{Q} < 1$ (we assume a common denominator) we construct a sample $\mathcal{S}_{\frac{p}{Q}}$ with empirical distribution $\hat{P}_{\frac{p}{Q}}$ such that for any $|h| \leq k$, the empirical distribution is uniform, and for $|h| = k+1$:

$$\hat{P}_{\frac{p}{Q}}(x_h) = \begin{cases} \left(1 + \frac{p_h}{Q\binom{n}{k+1}}\right) 2^{-|V|} & \text{If } \sum_{v \in h} x_v \text{ is odd} \\ \left(1 - \frac{p_h}{Q\binom{n}{k+1}}\right) 2^{-|V|} & \text{If } \sum_{v \in h} x_v \text{ is even} \end{cases} \tag{5.26}$$

Unlike the exact P_w, parities of larger sets might be very biased. However, these do not effect the resulting weights when searching for width k Markov networks.

We will build the sample as a pooling of $\binom{n}{k+1}$ equisized samples $\mathcal{S}^h_{\frac{p}{Q}}$, one for each candidate edge of size $k+1$. Each such $\mathcal{S}^h_{\frac{p}{Q}}$ will be constructed from Q equisized blocks of $(k+1)$-wise uniformly independent sample vectors. . But for p of these blocks, we will invert the elements of h appropriately so as to set the parity of x^t_h to be odd for all sample vectors in the block. Note that this can be done without disrupting the uniformity of any other set of vertices of size at most $k+1$. The resulting $\mathcal{S}^h_{\frac{p}{Q}}$ will be uniform on all subsets of size up to $k+1$, except for a bias of $\frac{p(h)}{Q}$ on h. Pooling these together yields the desired empirical distribution.

Using [AS91], $k+1$-wise independent blocks can be created of size $2n^{k+1}$, yielding a total sample size of $\binom{n}{k+1}Q2n^{k+1} = O(Qn^{2k+2})$, where Q is the common denominator of the rational weights.

Approximating the weights with rational biases

We now know how to construct a sample with specified *rational* biases. However, the biases corresponding to rational weights are not rational. We first show how to achieve approximate weights (i.e. with total error less then their granularity) with biases which are square roots of rationals, and then show how these can be approximated with actual rationals.

We saw in 5.25 that the biases of the mixture components should be approximately the square root of the desired weights. Using biases $r'(h) = \sqrt{w(h)}$ yields the following weights (where $b' = \frac{r'(h)}{\binom{n}{k+1}} < 1$):

$$\begin{aligned} w'(h) &= \sum_{i=2 \text{ even}} \frac{1}{i(i-1)} b'^i \\ &= \frac{b'^2}{2} + \sum_{i=4 \text{ even}} \frac{1}{i(i-1)} b'^i \\ &< \frac{b'^2}{2} + \sum_{i=4 \text{ even}} \frac{1}{i(i-1)} b'^4 \\ &= \frac{b'^2}{2} + \frac{\ln 4 - 1}{2} b'^4 \\ &= \frac{1}{2\binom{n}{k+1}^2} w(h) + e(h) \end{aligned} \tag{5.27}$$

$$\tag{5.28}$$

Where:

$$\begin{aligned} \sum_h |e(h)| &= \sum_h e(h) \\ &< \binom{n}{k+1} \frac{\ln 4 - 1}{2} \frac{(\max w)^2}{\binom{n}{k+1}^4} \\ &< \frac{0.19}{\binom{n}{k+1}^3} \max w^2 \end{aligned} \tag{5.29}$$

$$\tag{5.30}$$

Recall that we would like $\sum_h |e(h)| < \delta$ where δ is the smallest difference between weights. Since δ scales linearly with the weights, by scaling the weights down we can achieve this goal.

But since the weights might not be square rationals, taking their square root might produce irrational weights. This can be overcome in a similar fashion, by using a rational approximation to the square root.

5.4.3 The reduction and hardness

We saw how to reduce the maximum hypertree problem to the maximum likelihood Markov network problem, with the same k, and even if the variables are all binary. Note that our reduction is only pseudo-polynomial, as the sample size needed is polynomial in the value of the weights. However, since in Section 4.2 we show that the maximum hypertree problem is NP-hard, even with zero/one weights, this is enough to show NP-hardness of the maximum likelihood Markov network problem. The hardness result holds even for $k = 2$ (1-hypertrees are regular trees, for which the problem is tractable).

5.5 Approximating the Maximum Likelihood

In this thesis, we show that although finding the maximum weight bounded tree width graph is hard, an approximate solution can be found in polynomial time. That is, a graph with weight within a constant multiplicative factor of the optimal graph. We discuss how this type of approximation for the combinatorial problem translates into a sub-optimal solution for the maximum likelihood learning problem, as well as the general projection problem.

Recall the decomposition of the information divergence that was presented in Figure 5.3.2. When the target distribution is the empirical distribution, the negative cross entropy relative to it is exactly the log likelihood. Figure 5.3.2 can be viewed as representing the maximum log likelihoods of Markov networks over $\emptyset$ (i.e. fully independent models), Markov networks over G and and the maximum attainable log likelihood (the negative entropy of the empirical distribution). The weight of the graph is then the gain in maximum log likelihood versus the fully independent model. A constant factor approximation on the weight of the graph translates to a constant factor approximation on the gain in log likelihood.

As can be seen in Figure 5.3.2, a constant factor approximation for the weight of the graph does *not* provide for a constant factor approximation on the information divergence itself, but only for the reduction in information divergence relative to the fully independent model. Although approximating the likelihood itself is usually not of great interest (since it is usually of very high magnitude, and so even the fully independent model might be within a small factor of it), approximating the information divergence might be interesting, even when the target is an empirical distribution. For example, if the empirical distribution is "almost" a narrow Markov network, then approaching the optimal information divergence to within a small factor is much stringer then approximating the gain.

5.6 Discussion

The reductions we have shown between the maximum likelihood Markov network problem and the maximum hypertree problem are quite satisfactory in many ways. Both reductions are L-reductions, and are sufficient for studying approximation algorithms and hardness. Neither reduction, however, is strictly a polynomial reduction. Reducing Markov networks to the maximum hypertree problem produces a number of weights which is exponential in k, though the reduction is polynomial for a fixed width. The reverse reduction is only pseudo-polynomial. This pseudo-polynomiality does not prevent us from attaining hardness results, though it is interesting to see if the dependence on the value of the weights can be reduces.

Perhaps the more interesting gap is that we only show a reduction for the maximum hypertree problem with non-negative weights. Showing a reduction for monotone weight functions is an interesting open problem. Such a reduction is not necessary for showing the hardness results, but rather addresses a different interesting question: is the monotonicity constraint the true constraint on the weight function ? Or is there perhaps a stronger constraint that might aide in approximating the maximum hypertree.

Another interesting question is whether these techniques can be extended also to other measures of quality, beyond maximum likelihood, that incorporate in them also structural penalties.

Chapter 6

Windmills

Hypertrees are complex to work with and maximize. This is mostly due to the global nature of the characterization of hypertrees. There is no 'local' criterion for hypertrees or even hyperforests. By a local criterion we mean some constant-size property ψ over a finite number of vertices, such that a graph is a hypertree (or hyperforest) iff ψ holds for all subsets of vertices in $H(V)$.

As was discussed in Section 3.3, hypertrees can be defined using several criteria, involving decomposition, strong connectivity and the number of hyperedges. But all three of these properties are global and cannot be decomposed into local properties. We now introduce a simpler graph structure that can be used to capture much of the weight in hypertrees, and has a more local nature.

Let $T(V)$ be a rooted tree on the vertices V, with root r and depth at most k (i.e. the longest path beginning at r is of length k edges). The tree $T(V)$ defines the following hierarchy of vertices: r is at level zero. For any other vertex $v \in V$, consider the path from r to v. Vertex v is said to be on the *level* equal to the edge-length of the path.

Definition 6.1 (Windmill). *A k-*windmill *based on a rooted tree $T(V)$ with depth at most k is a hypergraph whose edges are the paths radiating from r in T, i.e.*

$$H(V) = \{h_v = \{r, \dots, v\} \textit{ is a path in } T\}.$$

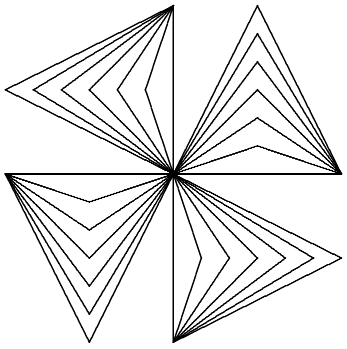

Figure 6-1: A 2-windmill

If all maximal paths radiating from r in T are of length k edges, then H is said to be a regular k-windmill.

A k-windmill is a hyperforest with tree-structure $T(H)$ where the hyperedges h_v replace each vertex v in T, i.e. $T(H) = \{\{h_v, h_{v'}\}|\{v, v'\} \in T\}$. Its Graham reduction follows the Graham reduction of leaves of the tree T. In particular, a k-windmill has treewidth at most k.

1-windmills are star graphs, and in some ways windmills are hyper-generalizations of star-graphs. Figure 6 shows a 2-windmill (which resemble physical windmills). Note that in a weighted 1-tree, there is always a disjoint set of stars that captures at least half of the weight of the tree. We will show that this can be generalized also for hypertrees.

Definition 6.2 (Windmill Farm). *A* k-windmill-farm *is a hypergraph that is a disjoint collection of k-windmills.*

Since each windmill is a hyperforest of width at most k, a windmill-farm is also a hyperforest of width at most k.

6.1 The Windmill Cover Theorem

In this section, we show that a hyperforest always has a windmill farm that captures a certain fraction of its weight. For simplicity, we first concentrate on the case of non-negative weights.

Theorem 6.1 (Windmill Cover Theorem). *For any hyperforest $H(V)$ of width k and non-negative weight function $w(\cdot)$, there exists a k-windmill-farm $F(V)$ such that $w(H) \leq (k+1)!w(F)$.*

Proof. We use a labelling scheme followed by a random selection scheme in which each hyperedge "survives" to be included in the windmill with probability at least $1/(k+1)!$. This means the total expected surviving weight is at least $w(F)/(k+1)!$, as desired. We then show that the surviving edges form a windmill.

The scheme is based on a $(k+1)$-coloring of the vertices, such that no two vertices in the same hyperedge have the same color. The existence of such a coloring can be proven by induction on the Graham reduction of the hyperforest: Let $H(V)$ be a hyperforest with leaf v, and recursively color $H(V-v)$. The leaf v has at most k neighbors (other members of its unique maximal edge) in $H(V-v)$, leaving a color available for v. This inductive proof specifies an order in which the vertices get colored. This is the reverse of the order in which vertices were Graham reduced. The order of coloring imposes on each hyperedge a (possibly partial) permutation of the colors used—namely, the order in which those colors were applied to vertices of the hyperedge.

From this ordering we construct our windmill farm. Choose a random permutation (ordering) π of the colors. We define a windmill farm F_π to contain all hyperedges whose color permutation (ordering) is consistent with π. For hyperedges with $k+1$ vertices, consistent simply means equal; for a hyperedge with fewer vertices, consistent means that the colors that *do* appear in the hyperedge form a prefix of the permutation π.

The permutation π of colors can be interpreted as a mapping between the colors and the $k+1$ levels of the windmill-farm F_π; each vertex now goes to the level of its color.

Each vertex of the first color $\pi(1)$ is the root of a windmill. Each vertex v of color $\pi(i+1)$ is at level i, with its parent being the vertex colored $\pi(i)$ in v's twig (the unique maximal hyperedge containing v when v was removed). Note that if the twig does not have a vertex of color $\pi(i)$ then no hyperedge containing v is in F_π: if $v \in h \in F_\pi$, then the partial color permutation imposed on h is at least an $i+1$-prefix of π and so must have a vertex u colored $\pi(i)$ which was colored before v. But if u was colored before v, then it was reduced after v, so it should appear in v's twig.

To show that F_π is indeed a windmill-farm over this tree structure, it is enough to show that for every $v \in h \in F_\pi$ of color $\pi(i+1)$, the vertex of color $\pi(i)$ in h is the parent of v in the windmill's rooted tree. Since the permutation of h agrees with π, a vertex u of color $\pi(i)$ exists in h and is colored before v. The vertex u is thus in v's twig, and so is v's parent.

The windmill-farm F_π might cover additional edges that were not explicitly selected by the scheme above, but since these have non-negative weight, the weight is at least the weight of the edges selected. A hyperedge of size r is selected to be in F_π if it is consistent with the permutation; this happens with probability $(k+1-r)!/(k+1)! \geq 1/(k+1)!$. Since the weight of edges is non-negative, the expected value contributed by any edge of weight w to F_π is at least $w/(k+1)!$. □

In fact, windmills can achieve the $1/d!$ approximation "simultaneously" for every edge of size d:

Corollary 6.2. *For any hyperforest $H(V)$ of width k, and non-negative weight function w, let w_d be the total weight of hyperedges of size d (so that the total weight of the hypertree is $\sum w_d$). Then there exists a k-windmill-farm contained in H of weight at least $\sum w_d/d!$*

Proof. We perform the above coloring and random selection, but include an edge in F_π if its colors appear in the same order in π, as a prefix or as an arbitrary subsequence. Then the probability that we include an edge of d vertices is $1/d!$. The parent of v of color $\pi(i+1)$ is selected to be the vertex in v's twig of color $\pi(j)$, for the maximum $j \leq i$, for which the twig includes such a vertex.

Note that under this selection criterion, F_π does not cover any additional edges not explicitly selected, and so $\mathbf{E}\left[w(F_\pi)\right] = \sum w_d/d!$ exactly. $\square$

Recall that we are actually interested in weight functions that are not necessarily non-negative, but rather are monotone on hyperforests. Even for such weight functions, a $1/(k+1)!$ fraction can still be achieved:

Corollary 6.3. *For any hyperforest $H(V)$ of width k and monotone weight function $w(\cdot)$, there exists a k-windmill-farm $F(V)$ such that $w(H) \leq (k+1)!w(F)$.*

Proof. Perform the same selection process as in Theorem 6.1, but analyze the weight of the resulting windmill farm differently. Instead of considering the weights of individual edges, consider the weight $g(v)$ gained when un-reducing v. That is, the difference in weight of the hyperforests before and after reducing v. Since every edge will be "gained" at exactly one reduction, $\sum_v g(v) = w(H)$. Furthermore, the gain is a difference in weight between two hyperforests, and so non-negative.

To analyze the expected weight of F_π, start from an empty hypergraph and add vertices according to their coloring (reverse reduction) order, keeping track of the weight of the sub-windmill-farm induced by F_π on vertices colored so far. Each colored vertex adds some non-negative gain. If the color permutation of a vertex's twig is a prefix of π, then the complete twig and all its subedges are covered by the farm, and the gained weight is exactly $g(v)$. Since this happens with probability at least $1/(k+1)!$, $\mathbf{E}\left[F_\pi\right] \geq w(F)/(k+1)!$. $\square$

In Chapter 7 we will devise an approximation algorithm for finding a maximum weight windmill farm, and use the above result to infer that the maximum weight windmill farm, which is a hyperforest, is competitive relative to the maximum weight hypertree.

6.2 Hardness of Approximation of Windmill

We now show that the maximum windmill forest problem is also NP-hard. In fact, we show that it is max-SNP-hard, implying that there exists some $\delta > 0$ such that it is NP-hard to find a windmill forest with weight within a multiplicative factor $1 + \delta$ from the maximum windmill forest. Unfortunately, this does not imply any hardness of approximation for the maximum hypertree problem.

Theorem 6.4. *For fixed $k > 1$, the maximum weight k-windmill problem (and even the maximal k-windmill problem for unit weights) is max-SNP hard.*

Proof. A reduction from max-2SAT □

6.3 Discussion

We introduced a family of graphs, windmill farms, and showed that there always exists such a windmill farm that captures $1/(k+1)!$ of the weight of a hyperforest. Is this figure tight, or is the true approximation ratio of windmill farms to hyperforests tight ? For $k = 1$ the ratio is indeed tight, but the answer is unknown for wider hyperforests. Note that even for $k = 1$, a weighted tree is necessary in order to show the tightness, as for uniformly weighted trees there is always a disjoint set of stars that captures $2/3$ of the weight.

Another interesting problem is using the hardness results on windmills to show hardness of approximation for hypertrees.

Chapter 7

Approximation Algorithms

In this chapter we present approximation algorithms for the maximum windmill-farm problem. As a consequence of the Windmill Cover Theorem (Theorem 6.1), these algorithms are also approximation algorithms for the maximum-hypertree problem.

In order to gradually introduce the reader to our approximation techniques, we first present, in Section 7.1, an approximation algorithm for a restricted case of the maximum 2-windmill-farm problem. In particular, we give an integer program whose solution is the maximum 2-windmill-farm. We then show how to round the solution of the corresponding relaxed linear program. In Section 7.2 we generalize the techniques and describe an algorithm for a windmill-farm of any width.

7.1 An Approximation Algorithm for 2-Windmills

In this section, we present some of our basic ideas in an algorithm for the 2-windmill problem. Recall that a 2-windmill is based on a tree with a root, a child layer, and a grandchild layer. We assume that there are weights only on triplets (not pairs or singletons), but this assumption can be made w.l.o.g. by adding an additional vertex u_{v_1,v_2} for every pair (v_1, v_2) and setting $w(v_1, v_2, u_{v_1,v_2}) \doteq w(v_1, v_2)$ while all other weights involving the new vertex u_{v_1,v_2} are set to zero.

7.1.1 Guessing Levels

For simplicity, we reduce to the case where the level of each vertex (root, child, or grandchild) is fixed. We do so by assigning each vertex to a random level (probability 1/3 for each). Any triple that appears in order v_1, v_2, v_3 in the optimal solutions will have its 3 vertices given the correct layers with probability $(1/3)^3 = 1/27$. Thus in expectation at least $1/27$ of the weight of the optimum solution will fit the random level assignment, so we expect there will be a solution that obeys the level assignment and has 1/27 of the optimum weight.

7.1.2 An Integer Program

Given the levels, we specify an integer linear program corresponding to the maximum 2-windmill problem. The variables in the IP are as follows:

- A variable x_{v_1,v_2} for every first-level node v_1 and second-level node v_2, which will be set to 1 if v_2 is a child of v_1.
- A variable x_{v_1,v_2,v_3} for every triplet of first-, second- and third-level nodes, respectively, which will be set to 1 if v_3 is a child of v_2, *and* v_2 is a child of v_1.

The integer program is then:

$$\max \sum_{v_1,v_2,v_3} x_{v_1,v_2,v_3} w_{v_1,v_2,v_3}$$

$$\begin{aligned}
(\forall v_2) \quad \sum_{v_1} x_{v_1,v_2} &= 1 \\
(\forall v_3) \quad \sum_{v_1,v_2} x_{v_1,v_2,v_3} &= 1 \\
(\forall v_1, v_2, v_3) \quad x_{v_1,v_2,v_3} &\leq x_{v_1,v_2} \\
(\forall v_1, v_2, v_3) \quad x_{v_1,v_2,v_3} &\geq 0 \\
(\forall v_1, v_2) \quad x_{v_1,v_2} &\geq 0
\end{aligned}$$

The first two equalities specify that each vertex is only on one path from the root, and the first inequality specifies that v_3 cannot be on path v_1, v_2, v_3 unless v_2 is descended from v_1—we will refer to this as requiring consistency of the paths.

Solving an integer program is not a straight-forward task. However, if we relax the requirement that the variables be integers, and allow them to be fractional numbers, we get a linear program, described by the same inequalities. All feasible solutions to the integer program are also feasible solutions to the linear program, and so the optimal solution to the linear program is at least as good the solution to the integer program. Such an optimal fractional solution (to a linear program) can be found in polynomial time. We would now like to round the fractional solution, without loosing too much of its value.

7.1.3 Rounding

We now show how to round a fractional solution, giving up a factor of less than 2 in the objective function value. Our rounding uses the natural probability distribution arising from the LP constraint that $\sum_{v_1} x_{v_1,v_2} = 1$; this suggests that v_2 can choose a parent vertex by selecting v_1 with probability x_{v_1,v_2}. However, this does not show how to choose parents for the third level vertices. We will, however, show that a simple two-step process works: first we round the second-level vertices, and then we let each third-level vertex make a greedy choice based on the previous level's rounding.

More precisely, the rounding to an IP solution $\tilde{x}$ from an LP solution x will be performed in two steps:

- For each v_2, assign one $\tilde{x}_{v_1,v_2} = 1$ at random according to the distribution given by x_{v_1,v_2}. The rest will receive value zero.
- For each v_3, assign one $\tilde{x}_{v_1,v_2,v_3} = 1$ with the maximum w_{v_1,v_2,v_3} among those (v_1, v_2) for which $\tilde{x}_{v_1,v_2} = 1$. The rest will receive value zero.

Note that the above rounding outputs a feasible IP solution. To analyze its value, we will consider each third-level vertex, and its contribution to the integer solution value, separately.

Lemma 7.1. *Consider a set of items such that item i has weight w_i. Suppose that each item i becomes "active" independently with probability p_i where $\sum p_i \leq 1$. Let W be the maximum weight of any active item. Then*

$$E[W] \geq (1/2) \sum w_i p_i$$

Proof. By induction. Assume there are n weights ordered such that $w_0 \geq w_1 \geq \cdots w_n$. Note that with probability p_0 item 0 becomes active and we get $W = w_0$, while with probability $1-p_0$ we get the maximum of the "subproblem" involving the remaining items. By induction, the expected maximum active weight not including item 0 has value at least $(1/2) \sum_{i>0} w_i p_i$. Observe also that $\sum_{i>0} w_i p_i$ is (at worst, since $\sum p_i \leq 1$) a weighted average of items less than w_0, so has value at most w_0. It follows that

$$\begin{aligned} E[W] &= p_0 w_0 + (1-p_0)(1/2) \sum_{i>0} p_i w_i \\ &= p_0 w_0 + (1/2) \sum_{i>0} p_i w_i - p_0 (1/2) \sum_{i>0} p_i w_i \\ &\geq p_0 w_0 + (1/2) \sum_{i>0} p_i w_i - p_0 (1/2) \left(\sum_{i>0} p_i \right) w_0 \\ &= (1/2) p_0 w_0 + (1/2) \sum_{i>0} p_i w_i \end{aligned}$$

as claimed. □

This lemma can be applied to our rounding scheme. Fix a particular third-level vertex v_3. Its contribution to the fractional LP objective value is $\sum_{v_1,v_2} x_{v_1,v_2,v_3} w_{v_1,v_2,v_3}$. Now consider the rounding step. Vertex v_3 is permitted to choose parent pair (v_1, v_2), contributing weight w_{v_1,v_2,v_3} to the objective, if v_2 chooses parent v_1, which happens with probability $x_{v_1,v_2} \geq x_{v_1,v_2,v_3}$. This almost fits the framework of the lemma with the variables

p_i set to x_{v_1,v_2,v_3}. There are two differences but they only help us. First, we may have $x_{v_1,v_2} > x_{v_1,v_2,v_3}$; however, this can only increase the odds of choosing a large weight. Second, the variables x are not independent. However, they are negatively correlated: the failure to choose some pair v_1, v_2 can only *i*ncrease the chance that we instead choose some other pair. This again only increases the expected contribution above the independent case. It follows from the lemma that we expect a contribution of at least $\sum w_{v_1,v_2,v_3} x_{v_1,v_2,v_3}/2$ from vertex v_3.

This analysis holds for all third-level variables, and combining over all of them yields an approximation ratio of 2 between the rounded solution and the LP solution. The weight of the farm is thus:

$$\begin{aligned}
w(\text{rounded IP farm}) &\geq w(\text{LP fractional farm})/2 \\
&\geq w(\text{maximal farm conforming to imposed levels})/2 \\
&\geq w(\text{maximal farm})/27/2 \\
&\geq w(\text{maximal hypertree})/6/27/2 = w(\text{maximal tree})/324
\end{aligned}$$

7.2 The General Case

Now we turn to the algorithm for general treewidth. We formulate a more general integer program, for any width k, and weights which are monotone on cliques, which does not assume that the assignment to levels is known. Then we give a more general rounding scheme—one that essentially applies the technique of the previous section one layer at a time. Some care must be taken to re-optimize the LP after each layer is rounded so that rounding can be applied to the next layer.

7.2.1 A General Integer Program

Consider a variable x_p for each simple path in G of length between 1 and k. Setting x_p to one corresponds to having p as a rooted path in a tree corresponding to a windmill in the solution (in particular, the first node in x_p is a root). We use the notation $|p|$ for the length of (number of nodes in) a path and $p \cdot q$, or $p \cdot v$ to denote the concatenation of two paths, or a path and vertex.

The weight w_p of a path is the gain in weight of adding the last vertex of p to the windmill. That is, for $p = q \cdot v$, $w_p = \sum_{h \subseteq p} w(h) - \sum_{h \subseteq q} w(h)$. Since the weight function is monotone on cliques of size at most $k+1$, it follows that the weights of paths are non-negative.

We first investigate the following integer program for the problem:

$$\begin{aligned} &\max \sum_p x_p w_p \\ &(\forall p, v) \quad x_{p \cdot v} \leq x_p \\ &(\forall v) \quad \sum_q x_{q \cdot v} \leq 1 \\ &(\forall p) \quad x_p \in \{0,1\} \end{aligned} \tag{7.1}$$

Both p in the first inequality and q in the second inequality vary over simple paths of length up to k, including the empty path. The first inequality requires consistency of paths, i.e. that every prefix of a path in the windmill is also in the windmill, as we did in the 2-windmill case. The second inequality constrains that there is only a single path from the root to any vertex, i.e. that paths do not converge, but rather form a tree.

We would now like to relax the integer program (7.1) to a linear program, so that we can solve the linear program and try to round the solutions. But instead of just relaxing the variables x_p to non-negative real values, we replace the two inequalities with a single unified inequality. The unified inequality agrees with (7.1) for zero-one values, but is stronger (more constrained) for fractional values. This constrains the fractional feasible polytope to be smaller, and so closer to the integral polytope, while still containing it. This reduces the integrality gap, and will aid us in rounding.

The linear program has a single equation for each simple path $p \cdot v$, $0 \leq |p| \leq k$. The variable x_ϵ (for the empty path of length 0) appears in the linear program only for uniformity of structure, but is set to one:

$$\begin{aligned} & \max \sum_p x_p w_p \\ (\forall p, v) \quad & \sum_q x_{p\cdot q\cdot v} \leq x_p \\ (\forall p) \quad & x_p \geq 0 \\ & x_\epsilon = 1 \end{aligned} \tag{7.2}$$

Both p and q in the inequality vary over simple paths of length up to k in the graph, including the empty path. Since we are only concerned with simple paths of length up to $k+1$, $v \in p$ is not allowed, and the sum is only over paths of length at most $k - |p|$ that are disjoint from $p \cdot v$. Note that since only simple paths of length up to $k+1$ have positive weight, allowing additional variables and equations for non-simple or longer paths will not affect the optimal solution.

The key constraint of (7.2) requires that the total fractional quantity of paths that share a prefix p and terminate at v is less than the fractional quantity of path p. This is a stronger inequality than the inequalities in (7.1):

- For any v and $|p| > 0$, since all variables are non-negative, and focusing on $q = \epsilon$, the inequality implies $x_{p\cdot v} \leq x_p$, the first inequality of (7.1).
- For $p = \epsilon$, we get $\sum_q x_{q\cdot v} \leq 1$, the second inequality of (7.1).

For integer values $\{0, 1\}$, there can only be a single path leading to each vertex. Thus for any p, v, there can only be one q with non-zero $x_{p\cdot q\cdot v}$, and so the inequality reduces to $x_{p\cdot q\cdot v} \leq x_p$, which follows from the path consistency. Therefore, on such values, (7.1) and (7.2) are equivalent.

7.2.2 A Rounding Scheme

Suppose now that we relax the integrality constraint and find a fractional solution. We propose to round each level iteratively, in a fashion similar to the previous section.

- Start with a solution x^0 to the LP, and no rounded variables $\tilde{x}$.
- For $i = 1$ to k:
  1. For each node v, the LP constrains that $\sum_p x^{i-1}_{p\cdot v} \le 1$. So choose a single path ending in v, or perhaps no path, such that each path $p \cdot v$ is chosen with probability $x^{i-1}_{p\cdot v}$. If p is of length $i-1$, set $\tilde{x}_{p\cdot v} \leftarrow 1$. In any case, for all other paths p of length $i-1$, set $\tilde{x}_{p\cdot v} \leftarrow 0$.
  2. Re-solve the LP, fixing all variables corresponding to paths of length up to i to be constants equal to their rounded values $\tilde{x}$. Take x^i to be the solution to this ith modified LP.

Note that since $\sum_{p:|p|=i-1} x^{i-1}_{p\cdot v}$ may be less than one, it may be that no path ending at vertex v will be rounded to 1 at some iteration. This corresponds to deciding that the vertex is at a higher level, or perhaps does not appear in the farm at all.

After the k iterations, only variables corresponding to length $k+1$ paths remain. The optimal solution to this LP is integral and can be found greedily, just as the last layer was found greedily in the 2-windmill algorithm.

This rounding method is a generalization of the rounding presented in the previous section for $k = 2$ and predetermined level assignments. The first iteration ($i = 1$) is trivial for predetermined levels, since all first-level vertices have only a single choice of ancestor (the unique level 0 vertex). The greedy assignment of the third level vertices in the second stage of rounding in the previous section exactly re-solves the linear program after rounding the second level nodes.

Note that the rounding step (1) itself preserves the expected value of the solution, but it might make the present solution infeasible. We show that after each step of rounding there is still a reasonably good feasible solution. To show this, we present an explicit solution to the i^{th} modified linear program.

Theorem 7.2. *The i^{th} rounding iteration decreases the optimum LP value by a factor of no worse than* $1/8(k+1-i)$.

Proof. At the ith iteration, consider the following solution $x^{(i)}$ to the modified LP:

- For each variable $x_{p\cdot q}$ with $|p| = i$, if $\tilde{x}_p = 0$, set $x^{(i)}_{p\cdot q} \leftarrow 0$ (this is mandated by the LP). If $\tilde{x}_p = 1$, set
$$x^{(i)}_{p\cdot q} \leftarrow \frac{x^{i-1}_{p\cdot q}}{4(k+1-i)x^{i-1}_p}. \tag{7.3}$$
- For each node v: if $\sum_p x^{(i)}_{p\cdot v} > 1$, then set all variables for paths in which v appears to zero (i.e. for all p, q set $x^{(i)}_{p\cdot v\cdot q} \leftarrow 0$). We say that the node *overflowed* and so all the paths it appeared on were *purged*.

We claim that the solution presented satisfies the LP constraints (since we purge any variables that violate the constraints) and that its value is at least $\frac{1}{8(k+1-i)}$ of the value before the iteration. The optimum LP value can only be better. Before proving this claim rigorously, let us present the intuitive reasoning for why $\frac{1}{8(k+1-i)}$ of the value is retained.

Consider a particular path $p \cdot q$, where $|p| = i$. The rounding scheme above rounds the prefix p to 1 with some probability α, and to 0 otherwise (also zeroing the path), but it also scales the path $p \cdot q$ by $1/4(k+1-i)\alpha$ if it does not zero it, so the *expected value* of $x_{p\cdot q}$ after rounding is just $x_{p\cdot q}/4(k+1-i)$. If that path has weight $w_{p\cdot q}$, we would hope that it continues to contribute a $1/4(k+1-i)$ fraction of its contribution to the starting solution. This will be true *unless* it is purged—that is, participates in some infeasible constraint. This happens if one of the vertices of q ends up with too large an "incoming value" on the fractional paths ending in it. To bound the probability of this event, conditioned on rounding x_p to one, we analyze the total incoming path-values into vertices of q. If this value is less than one, then surely no vertex in q overflows. We show that the expected value of this total, conditioned on rounding x_p to one, is less than half, and so with probability at least half there is no overflow.

To overcome the conditioning on rounding x_p, for each vertex v on q, we partition paths ending in v into those that share p as a prefix and those that do not. For those that do share p, the LP constraint for (p, v) guarantees an incoming value of at most x_p^{i-1} before scaling, and so $1/4(k+1-i)$ after scaling. For paths not sharing p, the conditioning just decreases the expected contribution, and the LP constraint for ϵ, v guarantees an expected total incoming value of at most $1/4(k+1-i)$ (after the scaling). Summing these two contributions over all $k+1-i$ vertices of q yields an expected total incoming value of one half. □

It follows by induction that the value of the (integer valued) LP optimum in the final step is no worse than $1/8^k k!$ times the original LP value. We therefore solve the windmill forest problem with an approximation ratio of $\frac{1}{8^k k!}$ and the hypertree problem with a ratio of $\frac{1}{8^k k!(k+1)!}$.

We return to proving that the explicit solution $x^{(i)}$ to the rounded LP, is in fact a feasible solution retaining $\frac{1}{8(k+1-i)}$ of the value, in expectation:

For any $i \geq 1$, let x^{i-1} be a solution of the $i-1$th linear program, and $x^{(i)}$ be the solution of the ith linear program as specified in Theorem 7.2. For uniformity and brevity of notation, we will include in these solutions also variables substituted as constants from $\tilde{x}$. We will show that $x^{(i)}$ is a feasible solution and that:

$$\mathbf{E}\left[\sum_p x_p^{(i)} w_p | x^{(i-1)}\right] \geq \sum_p x_p^{(i-1)} w_p \tag{7.4}$$

Feasibility of solution

We investigate the LP constraints for every p, v, and show that they are all satisfied:

- For $|p| > i$ and v, write $p = r \cdot s$ where $|r| = i$:

- If $\tilde{x}_r = 0$ or one of the vertices on p overflowed, then all variables in the constraint are nulled:

$$\sum_q x^{(i)}_{p\cdot q\cdot v} = 0 \leq 0 = x^{(i)}_p$$

- Otherwise, the constraint is a rescaled version the corresponding constraint in the $(i-1)$th LP, with some of the variables on the left-hand-side possibly nulled:

$$\begin{aligned}
\sum_q x^{(i)}_{p\cdot q\cdot v} &= \sum_q x^{(i)}_{r\cdot s\cdot q\cdot v} \\
&\leq \sum_q \frac{x^{i-1}_{r\cdot s\cdot q\dot{v}}}{4(k+1-i)x^{i-1}_r} \\
&\leq \frac{x^{i-1}_{r\cdot s}}{4(k+1-i)x^{i-1}_r} \\
&= x^{(i)}_{r\cdot s} = x^{(i)}_p
\end{aligned}$$

- For $p = \epsilon$, purging overflowing nodes guarantees that:

$$\sum_q x^{(i)}_{\epsilon\cdot q\cdot v} = \sum_q x^{(i)}_{q\cdot v} \leq 1 = x_\epsilon$$

- For $1 \leq |p| \leq i$: the value of x_p is already rounded, and so the constant $\tilde{x}_p$ appears in the LP.

  - If $\tilde{x}_p = 1$ then using the constraint on ϵ, v, which we already saw is satisfied:

$$\sum_q x^{(i)}_{p\cdot q\cdot v} \leq \sum_q x^{(i)}_{q\cdot v} \leq 1 = x_p$$

  - If $\tilde{x}_p = 0$ and $|p| < i$ then this constraint already appeared in the previous iteration's LP, and since zero-valued variables do not get their values increased in the rounding process, it is still true that:

$$\sum_q x^{(i)}_{p\cdot q\cdot v} = 0 = x_p$$

If $\tilde{x}_p = 0$ and $|p| = i$ then we have just set all $x^{(i)}_{p\cdot q}$ to zero and the above still holds.

Expected value of solution

We will show that for every p:

$$\mathbf{E}\left[x^{(i)}_p | x^{i-1}\right] \geq \frac{x^{i-1}_p}{8(k+1-i)} \tag{7.5}$$

For $|p| < i$, $x^{(i)}_p = x^{i-1}_p$ and (7.5) holds. For $|p| = i$:

$$\mathbf{E}\left[x^{(i)}_p | x^{i-1}\right] = \mathbf{E}\left[\tilde{x}_p | x^{i-1}\right] = x^{i-1}_p$$

and (7.5) holds.

We will denote by $x^{(i')}$ the value assigned to $x^{(i)}$ before possible purging, i.e. in the first step of the explicit rounding. All the expectations and probabilities in the following discussion are implicitly conditioned on x^{i-i}. We will analyze the expected value for any $x^{(i)}_{p\cdot q}$ such that $|p| = i$:

$$\begin{aligned}
\mathbf{E}\left[x^{(i)}_{p\cdot q}\right] &= \Pr(\tilde{x}_p = 1)\Pr(q \text{ is not purged}|\tilde{x}_p = 1)\mathbf{E}\left[x^{(i')}_{p\cdot q}|\tilde{x}_p = 1 \wedge q \text{ is not purged}\right] \\
&= x^{i-1}_p \Pr(q \text{ is not purged}|\tilde{x}_p = 1)\frac{x^{i-1}_{p\cdot q}}{4(k+1-i)x^{i-1}_p} \\
&= \frac{\Pr(q \text{ is not purged}|\tilde{x}_p = 1)x^{i-1}_{p\cdot q}}{4(k+1-i)} \qquad (7.6)
\end{aligned}$$

To bound the probability of purging q, consider a vertex $v \in q$, and analyze the expected value of paths ending in v, before checking for overflow. We will partition all such paths to paths which have p as a prefix, and paths which do not have p as a prefix. For paths that have p as a prefix:

$$\begin{aligned}
\sum_r \mathbf{E}\left[x^{(i')}_{p\cdot r\cdot v} | \tilde{x}_p = 1\right] &\leq \sum_r \frac{x^{i-1}_{p\cdot r\cdot v}}{4(k+1-i)x^{i-1}_p} && (7.7)\\
&= \frac{\sum_r x^{i-1}_{p\cdot r\cdot v}}{4(k+1-i)x^{i-1}_p}\\
&\leq \frac{x^(i-1)_p}{4(k+1-i)x^{i-1}_p} = \frac{1}{4(k+1-i)} && (7.8)
\end{aligned}$$

To bound (7.8) we used the linear program constraint for p, v.

For all other paths, not that conditioning on $\tilde{x}_p = 1$ can only decrease the probability of their prefix to be chosen in the rounding step:

$$\begin{aligned}
\sum_{s \text{ w/o prefix } p} \mathbf{E}\left[x^{(i')}_{s\cdot v} | \tilde{x}_p = 1\right] &\leq \sum_{s \text{ w/o prefix } p} \mathbf{E}\left[x^{(i')}_{s\cdot v}\right]\\
&\leq \sum_s \mathbf{E}\left[x^{(i')}_{s\cdot v}\right]\\
&= \sum_s \frac{x^{i-1}_{s\cdot v}}{4(k+1-i)}\\
&= \frac{\sum_s x^{i-1}_{s\cdot v}}{4(k+1-i)} \leq \frac{1}{4(k+1-i)} && (7.9)
\end{aligned}$$

The last inequality follows from the linear constraint for ϵ, v.

Combining (7.8) and (7.9) we get:

$$\mathbf{E}\left[\sum_t x^{(i')}_{t\cdot v} | \tilde{x}_p = 1\right] \leq \frac{2}{4(k+1-i)} = \frac{1}{2(k+1-i)}$$

To show that with reasonable probability *non* of the vertices in q overflow, we will sum the above over all vertices in q. $|p \cdot q| \leq k+1$ and $|p| = i$ and so $|q| \leq (k+1-i)$:

$$\mathbf{E}\left[\sum_{v\in q}\sum_{t} x_{t\cdot v}^{(i')}|\tilde{x}_p = 1\right] \leq \frac{k+1-i}{2(k+1-i)} = \frac{1}{2}$$

We can now use Markov's inequality to bound the probability of purging q:

$$\begin{aligned}\Pr\left(q \text{ is not purged}|\tilde{x}_p = 1\right) &= \Pr\left(\text{no vertex on } q \text{ overflows}|\tilde{x}_p = 1\right)\\ &\geq \Pr\left(\sum_{v\in q}\sum_{t} x_{t\cdot v}^{(i')} \leq 1|\tilde{x}_p = 1\right) \geq \frac{1}{2}\end{aligned}$$

Combining (7.6) and (7.10) to prove (7.5):

$$\begin{aligned}\mathbf{E}\left[x_{p\cdot q}^{(i)}\right] &= \Pr\left(q \text{ is not purged}|\tilde{x}_p = 1\right)\frac{x_{p\cdot q}^{i-1}}{4(k+1-i)}\\ &\geq \frac{1}{2}\frac{x_{p\cdot q}^{i-1}}{4(k+1-i)} = \frac{x_{p\cdot q}^{i-1}}{8(k+1-i)} \qquad (7.10)\end{aligned}$$

7.3 Discussion

We presented an integer program for the maximum windmill farm problem, and showed an iterative rounding algorithm for obtaining an integer solution from a fractional solution, retianing $\frac{1}{8^k k!(k+1)!}$ of the value of the fractional solution. Whether this ratio reflects the true integrality gap, or can be improved on, is an open question.

7.3.1 Iterative rounding

We suggested re-optimizing the LP after each iteration. But in the proof, a feasible (but not necessarily optimal) LP solution is explicitly constructed at each step. Thus, it is not technically necessary to re-solve the LP at each step—one can achieve the approximation ratio after just a single LP solution.

Note that there is a fundamental difference between the LP re-solving rounding, and the explicit rounding for which we prove the approximation ratio: The explicit rounding takes the optimal fractional solution x^0, and using only this solution, constructs an integer solution $\tilde{x}$. After initially solving the LP, the input to the problem (the weights) are no longer needed, and $\tilde{x}$ is determined solely by x^0. It follows that for any fractional solution to the LP, there exists an integer solution such that for *every* set of non-negative weights, it is within $\frac{1}{8^k k!}$ of the fractional solution. This is achieved by having $\mathbf{E}\left[\tilde{x}\right] = \frac{x^0}{8^k k!}$.

However, in the iterative LP re-solving method, the weights are used each time the LP is re-solved. The rounded solution $\tilde{x}$ might be different, even though the fractional optimum x^0 is identical. This is, in fact, what we did for the case $k = 2$, when the values for the third layer were rounded according to their weights, so as to maximize the weight. For $k = 2$, rounding the third level according to the values themselves, disregarding the weights, we would be able to prove an approximation ratio (for the IP) of only $1/4$, instead of the $1/2$ we achieved using the weights.

Is is conceivable that a better approximation ratio can be proven also in the general case, when the LP is properly re-solved at each iteration, using the knowledge of the precise objective function.

Chapter 8

Conclusions and Discussion

In this thesis, we have presented a graph algorithmic problem, the maximum hypertree problem, that is equivalent, in a rather strict sense, to the maximum likelihood Markov network problem. Hardness and approximability results on the maximum hypertree problem extend also to the maximum likelihood Markov network problem,

We believe that the maximum hypertree problem is an interesting and important combinatorial optimization problem, worthy of further study and analysis. We show how maximal hypertrees can be approximated by windmill-farms. We analyze the hardness of finding maximal windmill-farms, and present an approximation algorithm that achieves a constant approximation ratio for constant tree-width. But a wide gap remains between our hardness results for the maximum hypertree problem, and the approximation algorithm we suggest.

As was argued above, the exponential dependence of our algorithm's running time on the target tree-width k is unavoidable and non-problematic. However, an important open question is whether, given that we are willing to spend this much time, we can achieve an approximation factor that is a constant *independent* of k. We believe that the analysis of our algorithm's performance can be improved, but that the explicit rounding method will have an undesirable dependence on the tree-width. A direct analysis of the value of the iterative linear programs might yield a qualitative better approximation ratio.

Bibliography

[AP89] S. Arnborg and A. Proskurowski. Linear time algorithms for np-hard problems restircted to partial k-trees. *Disc. Appl. Math.*, 23:11–24, 1989.

[AS91] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. John Wiley & Sons, 1991.

[Bes74] Julian Besag. Spatial interaction and the statistical analysis of lattive systems. *Proceedings of the Royal Statistical Society, Series B*, pages 192–236, 1974.

[BGHK95] Hans L. Bodlaender, John R. Gilber, Hjalmtyr Hafsteinsson, and Ton Kloks. Approximationing treewidth, pathwidth, frontsize and shortest elimination tree. *Journal of Algorithms*, 18:238–255, 1995.

[Bod93] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1193.

[Bod96] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[Bod97] Hans L. Bodlaender. Treewidth: Algorithmic techniques and results. In Igor Privara and Peter Ruzicka, editors, *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science*, volume 1295 of *Lecture Notes in Computer Science*, pages 29–36, 1997.

[CFFK98] Gruia Călinescu, Cristina G. Fernandes, Ulrich Finkler, and Howard Karloff. A better approximation algorithm for finding planar subgraphs. *Journal of Algorithms*, 27(2):269–302, May 1998.

[CL68] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467, 1968.

[CLR89] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1989.

[Cou90] B. Courcelle. The monadic second-order logic of graphs i: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[CP87] S. Arnborg D. G. Corneil and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

[Das99] Sanjoy Dasgupta. Learning polytrees. In *Uncertainty in Artificial Intelligence*, 1999.

[DL93] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, March 1993.

[DL97] Paul Dagum and Michael Luby. An optimal approximation algorithm for Bayesian inference. *Artificial Intelligence*, 93(1–2):1–27, 1997.

[Jai98] K. Jain. A factor 2 approximation algorithm for the generalized steiner network problem. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 448–457, Los Alamitos, CA, November8–11 1998. IEEE Computer Society.

[KS01] David Karger and Nathan Srebro. Learning markov networks: Maximum bounded tree-width graphs. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, 2001. To appear.

[Mal91] Francesco M. Malvestuto. Approximating discrete probability distributions with decomposable models. *IEEE Transactions on systems, Man and Cybernetics*, 21(5):1287–1294, 1991.

[MP99] Marina Meila-Predoviciu. *Learning with Mixtures of Trees*. PhD thesis, Massachusetts Institute of Technology, 1999.

[Pea97] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann Publishers, revised second printing edition, 1997.

[RS95] N. Robertson and P. D. Seymour. Graph minors. xiii. the disjoint paths problem. *J. Comb. Theory Series B*, 63:65–110, 1995.

[SG97] Kirill Shoikhet and Dan Geiger. A practical algorithm for finding optimal triangulations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 185–190, 1997.

[STA97] David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 265–274, El Paso, Texas, 4–6 May 1997.

[WL83] N. Wermuth and S. Lauritzen. Graphical and recursive models of contingency tables. *Biometrika*, 72:537–552, 1983.