Learning Markov Networks: Maximum Bounded Tree-Width Graphs

David Karger* Nathan Srebro †

Abstract

Markov networks are a common class of graphical models used in machine learning. Such models use an undirected graph to capture dependency information among random variables in a joint probability distribution. Once one has chosen to use a Markov network model, one aims to choose the model that "best explains" the data that has been observed—this model can then be used to make predictions about future data.

We show that the problem of learning a maximum likelihood Markov network given certain observed data can be reduced to the problem of identifying a maximum weight low-treewidth graph under a given input weight function. We give the first constant factor approximation algorithm for this problem. More precisely, for any fixed treewidth objective k, we find a treewidth-k graph with an $f(k)$ fraction of the maximum possible weight of any treewidth-k graph.

1 Introduction

In this paper, we study a generalization of the maximum spanning tree problem: finding a maximum weight subgraph of bounded treewidth. This problem is motivated by a machine learning application: learning maximum-likelihood graphical models to fit data from an empirically sampled probability distribution. We show how our (NP-complete) graph problem arises naturally from this application. We then give the first approximation algorithms for the problem, achieving a polynomial-time constant-factor approximation for any fixed treewidth objective.

1.1 The Problem. One of the important areas of machine learning is the development and use of *probabilistic models* for classification and prediction. One popular probabilistic model [Pea97] is the class of *Markov networks*, which use a graph to represent dependencies among the variables in the probabilistic model. Given this graph, a particular probability distribution on the variables can be succinctly represented by specifying the (marginal) joint probability distribution over each set of variables that forms a clique.

In order to avoid over-fitting, it is important that the graph have no large cliques. At the same time, for efficient use of the model, the graph needs to be triangulated. 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.

In some applications, the graphical model is specified in advance. But in others, one begins with some observations and tries to find the graphical model that best fits these observations. 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. Sometimes, however, a higher treewidth is needed to fit the data well.

1.2 Our Results. We aim to learn, given some observed data, the best *treewidth-k* model of the data. We show that this problem reduces to a pure graph problem: given a candidate graph with weights on edges, and also on larger cliques of size up to $k+1$, find the maximum weight treewidth-k subgraph of the input graph. We show that for $k > 1$, this problem is NP-complete; even when only edges (and not larger cliques) are weighted. 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 is $n^{O(k)}$: a weight is specified for every candidate clique of size up to k (and in problems derived from machine learning, all these weights are usually nonzero). 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.

To the best of our knowledge, this is the first purely combinatorial formulation of the learning problem for general treewidth. It provides, for the first time, hardness results and provable approximation algorithms for the learning problem. The approximation algorithm is of a "global" nature, as opposed to local search heuristics which have been suggested before [Mal91].

Our approximation algorithm is based on two main ideas. The first is the identification of a structure called a k-*windmill*. While treewidth-k graphs can have complicated structure, k-windmills are easier to work with. We show that

*Research supported by NSF grant CCR-9624239 and a Packard Foundation Fellowship

†Research partially supported by NIH Genome Training Grant.
MIT Laboratory for Computer Science, Cambridge, MA 02139, USA. {karger, natis}@theory.lcs.mit.edu

any treewidth-k graph places at least a constant fraction of its weight in some k-windmill, and thus settle for approximating a maximum weight k-windmill. To find this windmill, we develop a linear-programming-based approximation algorithm. The linear program bears some 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 recent 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.

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

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 Nets*. Dagum and Luby 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]. In contrast, as we will see below, learning the distribution setting for a Markov network is trivial.

Most recent work on treewidth has been concerned with showing that some input 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 candidate edges, 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 candidate graph.

2 Maximum-Likelihood Dependency Graphs

In this section we introduce and motivate our problem. To do so we outline some concepts from machine learning that are not critical to understanding the results in this paper. This section provides the motivation for studying our problem, but can be skipped without loss of understanding of the algorithms.

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 conjectured distribution can be used to make predictions about future or partially observed data. Often, each observed data point is expressed as a vector of variables. A common approach in probabilistic machine learning is to assume that each data vector is drawn independently from some unknown probability distribution over possible vector values. One then aims to identify the probability distribution that best explains the observed data. The notion of "best explains" can be made concrete in many ways, but a common one is to choose the model that maximizes the likelihood (probability) of generating the observed data. This model is called the maximum-likelihood model.

In learning, one always faces a tradeoff between finding a model that fits the observed data and finding a model that generalizes well: the best model of the data is one which predicts that data and only that data, but is not usually useful for dealing with new data. To avoid *overfitting* the data, one can decide in advance on a limited class of possible models (probability distributions) and choose the model *from that class* that best fits the observed data. The model class should be complex enough to allow a decent approximation of the true distribution, yet not so complex as to cause overfitting. The complexity of a class is often quantified by the number of free parameters it allows.

For example, one very simple model class is one that assumes that the variables in the observed vector are *independent*, meaning that it is sufficient to develop a model for each variable separately and then take the product distribution. Such a strong assumption, however, might not allow good approximation when the true distribution contains important dependencies.

Thus, models are explored which attempt to capture a few important dependencies, yet are limited enough to avoid overfitting. *Markov networks* are a way to express such a dependency structure. In the discussion below, X is a random vector, and x is a possible sample value for X.

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

DEFINITION 2.2. (MARKOV NETWORK) *A random vector X_V, indexed by vertex set V, is a* Markov network *over an undirected graph $G(V)$ iff each random variable X_v, conditioned on its neighbors, is independent of all other elements of X_V:*

$$(2.1)\quad (\forall v \in V) \quad X_v \perp \{X_u \mid u \neq v, (u,v) \notin G\} \mid \{X_u | (v,u) \in G\}$$

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 complete graph: the independence requirement is satisfied vacuously. As a more interesting example, any finite-length Markov chain is a Markov network whose underlying graph is a path: each variable is directly dependent on only its predecessor and successor. Note that if X is a Markov network over G, it is also a Markov network over any supergraph of G: adding edges to the graph removes independence constraints and thus allows for a more general class of distributions. The sparser a graph, the more limited the class of Markov networks it models.

Of particular interest are Markov networks over triangulated graphs.

DEFINITION 2.3. *A graph is said to be* triangulated *iff it does not contain any minimal cycles of more than three vertices.*

Over a triangulated graph G, all marginal distributions, and therefore all conditional distributions, can be determined from the marginal distributions over each maximal clique in G. Given a stored representation of each of these clique marginals, one can compute all other marginals and conditional distributions in linear time—that is, linear in the size of the tables needed to hold the clique marginals, and hence exponential in the clique sizes. The number of free parameters for the Markov network is also roughly exponential in the clique sizes. It is thus desirable to bound the clique sizes, both to ensure feasible storage and computation-time requirements, and to limit the model complexity and avoid overfitting.

One possible scenario is that the graph G is specified in advance (e.g. from prior knowledge about the relationships between variables), and we limit ourselves to the class of Markov networks over G. In such a scenario, and if G is triangulated, the maximum likelihood Markov network $\tilde{X}$ over G can be explicitly calculated from the empirical distribution $\hat{P}$ (that is, the probability distribution in which each of the T observed sample points is given probability $1/T$, and all other sample points are given probability 0):

$$P_{\tilde{X}}(x) = \prod_{h \in \mathit{Clique}(G)} \phi_h(x_h) \tag{2.2}$$

where the *clique factors* ϕ_h are functions of the outcomes of random variables indexed by the clique $x_h = \{x_v | v \in h\}$, and are given by the following recursion:

$$\phi_h(x_h) = \frac{\hat{P}_h(x_h)}{\prod_{h' \subset h} \phi_{h'}(x_{h'})}. \tag{2.3}$$

The product in (2.2) and the recursion (2.3) are over all, not necessarily maximal, cliques. It is important to note that the factor ϕ_h for any clique h depends only on the empirical marginal distribution $\hat{P}_h$ over the clique, and is completely oblivious to the structure graph G, other than the fact that it contains h as a clique.

In this work, we investigate the situation in which the graph is unspecified, but required to be triangulated and with bounded clique size. We would like to find the maximum likelihood Markov network over any such graph. The core problem is finding the maximum likelihood graph itself (i.e. the graph over which the maximum likelihood Markov network is achieved), since the maximum likelihood distribution would then be specified by (2.2). We can associate a maximum likelihood with every candidate graph G—that is, the maximum, over all Markov networks over G, of the probability of observing the data under the model. We would like to find the triangulated, bounded clique-size graph G with the largest maximum-likelihood.

For any candidate triangulated graph G, (2.2) specifies the maximum likelihood Markov network over it, and thus lets us analyze the maximum likelihood $L(G)$ that G can achieve on the samples $\{x^t\}_{t=1}^T$. The probability the net generates the data is just the product of the probabilities that it generates each sample point, so

$$\begin{aligned}\log L(G) &= \log \prod_t \prod_{h \in \mathit{Clique}(G)} \phi_h(x_h^t) \\ &= \sum_{h \in \mathit{Clique}(G)} \sum_t \log \phi_h(x_h^t)\end{aligned}$$

we now plug in the optimum clique factors from (2.3) based on the empirical distribution:

$$\begin{aligned}&= \sum_{h \in \mathit{Clique}(G)} \sum_{x_h} T \cdot \hat{P}_h(x_h) \log \phi_h(x_h) \\ &= T \sum_{h \in \mathit{Clique}(G)} \mathbf{E}_{\hat{P}} \left[\log \phi_h(X_h)\right] \qquad (2.4)\end{aligned}$$

For each candidate clique h, the term $\mathbf{E}_{\hat{P}}[\phi_h(X_h)]$ in the sum depends only on the empirical marginal over h, and not on the rest of G. Consider a weight function over candidate cliques, such that $w(h) = \mathbf{E}_{\hat{P}}[\log \phi_h(X_h)]$. This weight function can be calculated based on the empirical distribution alone, using the explicit recursion:

$$\begin{aligned}w(h) &= \mathbf{E}_{\hat{P}}\left[\phi_h(X_h)\right] \\ &= \mathbf{E}_{\hat{P}}\left[\frac{\hat{P}(x_h)}{\prod_{h' \subset h} \phi_{h'}(x_{h'})}\right] \\ &= \mathbf{E}_{\hat{P}}\left[\log \hat{P} x_h\right] - \sum_{h' \subset h} \mathbf{E}_{\hat{P}}\left[\log \phi_{h'}(X_{h'})\right] \\ &= -H(\hat{P}(h)) - \sum_{h' \subset h} w(h') \qquad (2.5)\end{aligned}$$

where $H(\hat{P}(h))$ is the entropy of the empirical marginal.

Note that a single vertex always forms a clique, and so will always appear in the sum of (2.4). The contribution of these singleton cliques is exactly $\log L(\emptyset)$, the log maximum likelihood of the data over a fully independent model, where there are no edges, and all variables are independent. Separating out the singletons, and using our newly defined weights, we get:

$$(2.6)\quad \log L(G) = \log L(\emptyset) + T \cdot \sum_{h \in Clique(G), |h|>1} w(h)$$

Equation (2.6) expresses the maximum likelihood of a given candidate graph as a "gain" over a simple baseline model, in which all variables are independent. The "gain" is a simple sum of weights (derived from the empirical data) of all non-trivial cliques that appear in the graph. Maximizing the maximum-likelihood is thus equivalent to maximizing this "gain", so we can formulate the problem of finding the maximum likelihood Markov network as one of finding a triangulated graph of bounded clique size that maximizes the total (summed) weight of its (not only maximal) cliques. Note that the singleton weights are defined for the recursion (2.5) but are not summed in calculating the weight of a graph.

Note that the log maximum likelihood is a negative measure. The log maximum likelihood of the independent model, $\log L(\emptyset)$, is a negative quantity, and the gain is positive, which brings the log maximum likelihood of G closer to, but still usually far away from, zero. Log likelihoods are usually very large magnitude negative numbers, whose magnitude mostly reflects, and is bounded from above by, the inherent entropy of the distribution being sampled. Because of this, it is common to analyze the improvement in log-likelihood over some baseline model.

The approximation results presented in this paper are also of this nature. The weight of the graph G is the gain in maximum-log-likelihood, so by approximating the weight of the graph, we approximate the gain in maximum-log-likelihood.

Although the weight of a triangulated graph, being the gain in log maximum likelihood, will always be non-negative, the same cannot be said about individual weights. The existence of negative weights is a potential pit-fall in many combinatorial algorithms, including the ones presented here. However, the weights defined above (ignoring the singleton weights) do have the following property which enables our algorithms to work properly:

DEFINITION 2.4. (MONOTONE WEIGHTS) *A weight function* $w : \{h \subset V \mid |h| \leq d\} \rightarrow \Re$ *is said to be* monotone *if for any* $|h_1| \leq d$ *and* $h_2 \subset h_1$, $\sum_{h' \subset h_1} w(h') \geq \sum_{h' \subset h_2} w(h')$.

It is also interesting to note that for every non-negative weighting of potential cliques, there exists an empirical distribution that yields weights proportional to this set of weights [Sre00]. Thus the problem of finding a graph of maximum weight over a non-negative weight function can 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 values of the weights. Still, this is enough to show the NP-hardness of finding a maximum-likelihood triangulated Markov network of bounded clique size.

3 Problem definition

In the above section, we were able to reduce the maximum-likelihood Markov network problem to the following: given a weight for each candidate clique of size at most $k+1$, find a triangulated graph with cliques of size at most $k+1$ that maximizes the summed weight of its cliques. As will be defined below, the constraints on the graph specify that it should have *treewidth* at most k. We will work with a hypergraph view of tree width, which is similar to that discussed by Bodlaender [Bod93], and will search for a hypertree of width at most k that maximizes the weight of the hyperedges it covers.

3.1 Preliminaries: Graph and Hypergraphs. A *graph* $G(V)$ is a collection of unordered pairs (*edges*) of the *vertex* set V: $G(V) \subset \binom{V}{2}$. A *hypergraph* $H(V)$ is a collection of subsets (edges, or sometimes explicitly *hyper-edges*) 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 the set-theory notation and denote $h' \in H$ even if h' is only 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.

3.2 Acyclic Hypergraphs and Treewidth. Recall that an acyclic graph can be defined recursively as an empty graph, or a graph containing a *leaf*, i.e. a vertex incident on a single edge, such that removing the leaf yields an acyclic graph. We will use a similar definition for acyclic hypergraphs.

DEFINITION 3.1. *A vertex* v *of a hypergraph* $H(V)$ *is called a* leaf *if it appears only in a single maximal hyperedge* $h \in H$. *The hyperedge* h *is called the* twig *of* v.

DEFINITION 3.2. *A hypergraph* $H(V)$ *is said to be* acyclic *if it is empty, or if it contains a leaf* v *such that the induced hypergraph* $H(V - v)$ *is acyclic.*

An acyclic hypergraph is also referred to as a *hyperforest*. The implied iterative reduction process is called a *Graham reduction* of the hyperforest. The twig of a vertex of a hyperforest is taken to mean the twig of the vertex when it is reduced. Note that although the twig is a maximal hyperedge at the time of reduction, it might not be maximal in the hyperforest itself. If we consider the underlying triangulated graph of "normal" (2-endpoint) edges defined by the hyperforest, a Graham ordering is an ordering in which, as each vertex is eliminated, its neighbors form a clique.

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

Thus, the width of a standard tree is 1.

A hyperforest which is maximal among hyperforests of the same width is said to be a *hypertree*. Since we are concerned with maximum structures, we will be interested mostly in hypertrees.

DEFINITION 3.4. *A* tree decomposition *of a graph $G(V)$ is a covering hyperforest $H(V)\supseteq G(V)$.*

DEFINITION 3.5. *The* treewidth *of a graph is the width of the narrowest hyperforest covering it.*

The following theorem establishes the well-known connection between hyperforests and triangulated graphs:

DEFINITION 3.6. *The clique-hypergraph of a graph G is the hypergraph in which the hyperedges are the cliques of G.*

THEOREM 3.1. *A hypergraph is acyclic if and only if it is a clique hypergraph of some triangulated graph. A graph is triangulated if and only if its clique hypergraph is acyclic.*

COROLLARY 3.1. *For a graph G, the treewidth of G is equal to the minimum over triangulations G' of G, of the maximal clique size in G', minus one:*

$$width(G) = \min_{triang\, G'\supseteq G} \max |Clique(G')| - 1 \tag{3.7}$$

Many alternative equivalent definitions of hyperforests and treewidth appear in the literature. For a survey and details on the above definitions and results, see [Sre00].

3.3 The Maximal Hypertree Problem. As a consequence of Theorem 3.1, we can discuss weights of hyperforests instead of triangulated graphs with weights on cliques. 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, it is essential to assign weights also to larger candidate hyperedges. A hyper-weight function assigns a weight to candidate hyperedge of vertices of arbitrary size, and the weight of a hypergraph is the sum of the weights of edges *covered* by it: $w(H)=\sum_{h\in H} w(h)$. Recall (from Definition 2.4) that a hyper-weight function assigning weights to edges of size up to d is *monotone* if, for every $h_2\subset h_1, |h_1|\le d$, the weight of the hypergraph with maximal edge h_2 is less than or equal to the weight of the hypergraph with maximal edge h_1.

We are now ready to state the maximal hypertree problem:

> Given as inputs:
>
> - An integer treewidth k.
> - A vertex set V and monotone weight function $w : \binom{V}{\le k+1} \to \Re^+$ on hyperedges of size up to and including $k+1$.
>
> Find a hypertree $H(V)$ of treewidth at most k that maximizes $w(H)$.

Note that such a monotone weight function is also monotone on hyperforests of width at most k. Since every hyperforest has a covering hypertree, the maximal hyperforest can always be taken to be a hypertree. So this problem could have also been stated as finding a maximal hyperforest.

When $k=1$, the maximum hypertree problem is simply the maximum spanning tree problem.

4 Windmills

General graphs of small treewidth are difficult to work with. Therefore, we introduce a simpler graph structure that can be used to capture much of the weight in hypertrees. 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 has 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. Nodes on the path from r to v are its *ancestors*.

DEFINITION 4.1. *A k-*windmill *based on a rooted tree $T(V)$ is a hypertree whose maximal edges are the root-leaf paths in T.*

Such a k-windmill is a hypertree: its Graham reduction follows the (standard) Graham reduction of leaves of the tree T. 1-windmills are star graphs.

A *k-windmill-farm* is a hypergraph that is a disjoint collection of windmills. Since each windmill is a hypertree, a windmill-farm is a hyperforest.

THEOREM 4.1. (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 $w(H)\le(k+1)!w(F)$.*

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 ordering 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 a 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, and 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. 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 4.1. *For any hyperforest $H(V)$ of width k, 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 $\mathbf{E}[w(F_\pi)] = \sum w_d/d!$ exactly.

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

COROLLARY 4.2. *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 4.1, 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, the gained weight is exactly $g(v)$. Since this happens with probability at least $1/(k+1)!$, $\mathbf{E}[F_\pi] \geq w(F)/(k+1)!$.

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 hyper-forest, is competitive relative to the maximum weight hyper-tree.

4.1 Hardness Results. We present hardness results on windmill approximation. It is important to note that these do not extend to showing the hardness of approximation of maximal hypertrees. While we can show that finding a maximum hypertree, as well as a maximum likelihood Markov network, is NP-hard, our analysis does not even preclude the possibility of a PTAS. However, we conjecture that our hardness results do extend to the hypertree problem.

THEOREM 4.2. *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

A similar reduction can be used to show NP-hardness of the maximal hypertree problem for fixed $k > 1$ (even with unit weights, and weights only on edges of size 2). However, this reduction does not show hardness of approximation.

THEOREM 4.3. *For k part of the input, and in particular for some $k = \Omega(n)$, no polynomial time algorithm can find an $n^{1-\epsilon}$ approximation to the maximum k-windmill unless $P = NP$.*

Proof. A reduction from independent set, where each vertex is converted to a path, and paths cross (so cannot be in the same windmill) if the corresponding vertices are neighbors.

5 An Approximation Algorithm for 2-Windmills

In this section, we present some of our basic ideas in an algorithm for the 2-windmill problem with non-negative weights. Recall that a 2-windmill is 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.

5.1 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 optimal layers with probability $(1/3)^3 = 1/27$. Thus in expectation at least $1/27$ of the weight of the optimum solution will obey the level assignment, so there will be a solution that obeys the level assignment and has 1/27 of the optimum weight.

5.2 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 belongs to v_1 *and* to v_2 (that is, that v_3 is a child of v_2 which 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.

To get an approximation to the integer program, we first solve the relaxed linear program given by the above equations, and then perform the randomized rounding described below.

5.3 Rounding. We now show how to round a fractional solution, giving up a factor of less than 1.6 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 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 5.1. *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

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

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 $(1 - 1/e) \sum w_{v_1,v_2,v_3} x_{v_1,v_2,v_3}$ from vertex v_3.

This analysis holds for all third-level variables, and combining over all of them yields an approximation ratio of $1 - 1/e$ between the rounded solution and the LP solution.

The farm we found thus has an expected weight of at least $1/324$ of the weight of the maximal hypertree (a factor of $1/2$ for the rounding gap, 27 for the randomly assigned levels, and 6 for the gap between a windmill farm and the hypertree). A more careful analysis can improve the constant to 1/108.

6 The General Case

Now we turn to the algorithm for general treewidth. We formulate a more general integer program, for any width k, monotone weights, 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.

6.1 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 path in 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, it follows that the weights of paths are non-negative.

The linear program has a single equation for each such 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:

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

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 the LP requires that the total fractional quantity of paths that share a prefix p and terminate in v is less than the fractional quantity of path p. This is a stronger constraint than the ones that arise naturally in the *integer* linear program:

- For $p = \epsilon$, the inequality specifies that every vertex is at the end of only a single path.

- For any v and $|p| > 0$, since all variables are non-negative, and focusing on $q = \epsilon$, the inequality implies $x_{p \cdot v} \leq x_p$, requiring the consistency of paths as we did in the 2-windmill case.

The more complex sum over nonempty paths q is taken to further limit the fractional polytope so as to reduce the integrality gap and aid in rounding.

6.2 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 $\sum_p x_{p \cdot v}^{i-1} \leq 1$. So choose a single path p with probability $x_{p,v}^{i-1}$. If this path 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 $\sum_p x_{p \cdot v}^{i-1}$ may be less than one, it may be that no path ending in some vertex v will be rounded to 1 at some iteration. This corresponds to deciding that the vertex is at a higher level.

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 optimal 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 6.1. *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 $x_{p \cdot q}^{(i)} \leftarrow 0$ (this is mandated by the LP). If $\tilde{x}_p = 1$, set

$$x_{p \cdot q}^{(i)} \leftarrow \frac{x_{p \cdot q}^{i-1}}{4(k + 1 - i)x_p^{i-1}}. \tag{6.8}$$

- For each node v: if $\sum_p x_{p \cdot v}^{(i)} > 1$, then set all variables in which v appears to zero (i.e. for all p, q set $x_{p \cdot v \cdot q}^{(i)} \leftarrow 0$). We say that the node *overflowed* and so all the paths it appeared on were *purged*.

The solution presented satisfies the LP constraints (since we purge any variables that violate the constraints). We claim 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. A complete proof can be found in [Sre00]; here we provide an outline.

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 follow 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 $\frac{1}{8^k k!(k+1)!}$.

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. Furthermore, this explicitly re-solution does not directly depend on the weights w_p, but only on the value of the fractional solution. Of course, using the weights (as we did in the previous section for $k = 2$) and re-solving the LP at each step seems likely to yield a better solution in practice.

7 Conclusions

We present an interesting graph algorithmic problem, finding maximum hypertrees, that has important applications in unsupervised machine learning. 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.

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 qualitatively better approximation ratio.

An interesting question about the structure of the windmill polytope arises from the discussion at the end of section 6.2. Is the integrality gap that can be guaranteed for any specific objective function smaller than the integrality gap that can be guaranteed independent of the weights?

8 Acknowledgments

Work on this problem originated in a class taught by Tommi Jaakkola. We are grateful to him for introducing us to Markov networks and for many helpful discussions on the topic.

References

[BFMY83] Catriel Beery, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J of the ACM*, 30(3):479–513, 1983.

[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.

[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.

[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.

[Gra79] M. H. Graham. On the universal relation. Technical report, University of Toronto, 1979.

[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.

[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.

[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.

[Sre00] Nathan Srebro. Maximum likelihood Markov networks: An algorithmic approach. Master's thesis, Massachusetts Institute of Technology, 2000.

[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.