Learning Bounded Treewidth Markov Networks: A Survey

Harr Chen Boon Thye Thomas Yeo
6.856 Randomized Algorithms

1 Introduction

Graphical models are a powerful tool in statistical machine learning. They are used to represent the properties of a joint distribution over a set of random variables—in particular, they encode conditional independencies between the variables. A particular kind of graphical model is the *Markov network*, where the variables are represented as vertices and dependencies are encoded as *undirected* edges between the vertices.

A natural question that arises in the study of Markov networks is how to learn the model from observation data. This in turn is divided into two problems—learning the graph structure, and inferring the conditional distributions at each vertex. The latter problem is known as the *inference* problem, and is well studied [8]. The former problem of learning the graph structure itself, is what we study in this report.

Learning graph structure is an exercise in trading off between representative power and overfitting. A fully connected graph on the vertices will always fit any set of observation data best (in terms of training error), but does not make any independence assumptions and thus will almost always overfit and is slow. Therefore, it is useful to learn from a restricted hypothesis space of graph structures.

In this report, we look at two papers from the literature that operate on the same restricted space of models—specifically, so-called *bounded treewidth* triangulated graphs. They solve closely related problems. Karger and Srebro (hereafter KS) look at learning the *maximum likelihood* bounded treewidth graph given some observation data [3, 7]. They show that the problem is NP-complete, but use an integer linear program, relaxed to a linear program, and a clever iterative rounding scheme to derive a randomized approximation algorithm that is optimal to within a constant factor. Narasimhan and Bilmes (hereafter NB) demonstrate an algorithm to efficiently *PAC-learn* a bounded treewidth graphical model [5]. That is, they show that, assuming the underlying source of the observation data is a bounded treewidth graphical model, we can efficiently recover a bounded treewidth model that is close, with respect to KL-divergence, to the true model. "Close" means within an arbitrarily small ϵ-factor with arbitrarily high confidence $1 - \delta$, and "efficient" means in time polynomial in n, $1/\epsilon$, and $1/\delta$ (assuming treewidth is a constant).

We can understand the difference between the approaches through KL-divergence. KS's goal is to minimize the KL-divergence between the *empirical* distribution induced by the observation data, and the resulting estimated maximum likelihood model. It takes as input a previously generated observation sequence, and operates only on that data. NB tries to minimize the KL-divergence between the *true* distribution and the resulting estimated distribution. It relies on being able to dynamically sample from the true distribution, and it will usually only return a valid answer if the true distribution is of the right type.

This paper is organized as follows. In sections 2 and 3, we introduce some graph and machine learning concepts common to both KS's and NB's work. We then study each individually, with an emphasis on the randomization techniques and concepts used.

In section 4, we will look at KS's reduction of the maximum likelihood Markov network to a purely algorithmic problem—the *maximum hypertree* problem. This problem can be approximated by the so-called maximum *windmill farm* problem (section 5). Next, they formulate an integer linear program to solve the maximum windmill farm (section 6), relax it to a regular linear program (section 7), and present a sophisticated rounding scheme that preserves a constant integrality gap (section 8).

We then look at NB's approach to PAC-learning a bounded treewidth model. They show that mutual information, a measure of the closeness of two probability distributions, is a so-called *submodular and symmetric* function (section 10). This lends itself to a polynomial time minimization technique. Then they show that finding the minimum mutual information based on the sampled distribution, while not strictly submodular, is guaranteed to be within an ϵ-factor of the true minimum with constant probability. This allows us to partition the variables into sets that have low mutual information with respect to each other (section 11), and therefore admits a polynomial time algorithm that recovers a graphical model that is arbitrarily close to the true model, as measured by KL-divergence (section 12).

2 Graphical Models and More

A Markov network consists of a tuple (G, P), where G is an undirected graph and P is a joint probability distribution. The vertices of G represents the random variables of P and the absence of edges in G represents conditional independence between the random variables of P.

Definition 2.1 (Separator). *A separator $S \subset V$ is a set of vertices whose removal disconnects the graph.*

Hence in figure 1(a), $\{2, 4\}$ is a separator and so is $\{5, 7\}$. In figure 1(b), $\{5\}$ is a separator. In a Markov network, if S separates the network into a collection of disjoint subgraphs, $\{A_i\}$, then this implies that the A_i's are independent given S. This further implies that A_j and A_k are independent given S, for $j \neq k$.

Definition 2.2 (Triangulation). *A graph G is triangulated if every cycle of length at least 4 has a chord.*

Figure 1(b) shows a triangulated graph, while the graph in figure 1(a) is not triangulated because of the cycle $\{2, 5, 6, 7, 4\}$. Using the definition of separators, another alternative definition of triangulation is as follows:

Definition 2.3 (Triangulation). *A graph G is triangulated if all minimal separators are cliques. A clique is a set of vertices that are completely connected. A separator, S is minimal if there does not exist a proper subset of S that separates G.*

Hence in figure 1(a), $\{5, 7\}$ is a minimal separator but is clearly not a clique, whereas $\{2, 4\}$ is both a minimal separator and a clique. The existence of minimal separator $\{5, 7\}$ means figure 1(a) is not triangulated. that We generalize the concept of a graph to a hypergraph:

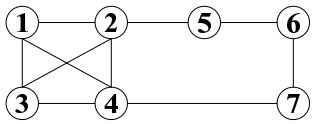
Definition 2.4 (Hypergraph). *A hypergraph $H(V)$ is a collection of subsets of vertices ($hyperedges$) of the vertex set V. Furthermore, for all hyperedges, $h \in H$, $h' \subset h$ implies $h' \in H$, i.e. a hypergraph includes all subsets of its edges.*

For example, see figure 1(a).

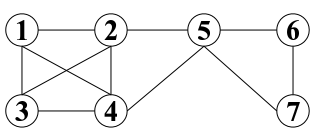
A clique-hypergraph of a graph G is the hypergraph whose hyperedges are the cliques of G. *For the purpose of this paper only, we will only consider clique-hypergraphs*, hence an alternative definition of hypergraph is the following:

Definition 2.5 (Hypergraph). *A hypergraph $H(V)$ is a graph $G(V)$ whose cliques are hyperedges of $H(V)$. Hence in this paper, we use cliques and hyperedges interchangeably.*

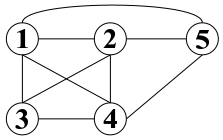
We now define leaves and acyclicity in hypergraphs:



(a) Hypergraph



(b) Hyperforest



(c) 3-Hypertree

Figure 1: Hypergraph, Hyperforest, Tree-Decomposition and 3-Hypertree

Definition 2.6 (Leaf). *A maximal hyperedge $h \in H$ is one where there does not exist $h' \supset h$, such that $h' \in H$. A leaf of a hypergraph $H(V)$ is a vertex that appears in only one maximal hyperedge, $h \in H$.*

Hence in figure 1(b), vertices 1, 3, 6 and 7 are leaves, while 2, 4, 5 are not.

Definition 2.7 (Hyperforest). *A hypergraph $H(V)$ is acyclic if it is empty or it contains a leaf, v such that $H(V - v)$ is acyclic. A hyperforest is an acyclic hypergraph.*

Because of our paper-specific alternative definition of the hypergraph, we have the following fact: a hyperforest $\equiv$ triangulated graph. This recursive definition of acyclicity induces a Graham reduction of the hyperforest: we begin with a hypergraph, $H(V)$ and iteratively remove the leaves of $H(V)$ until there are no more leaves. If the resulting hypergraph is empty, we conclude $H(V)$ is a hyperforest. Note that the Graham reduction is usually not unique, since there might be multiple candidate leaves that can be removed at each stage of the reduction. For example, for the hyperforest in figure 1(b): we can first remove vertices 1, 3, 6 and 7, and then remove 2, 4 and 5. Alternatively, we can remove vertices 1 and 3, followed by 2 and 4, followed by 5, 6 and 7.

We now define the concept of treewidth.

Definition 2.8 (Treewidth). *The treewidth of a hyperforest $H(V)$ is the size of its largest hyperedge minus 1:* $max_{h \in H}|h| - 1$.

In the context of our paper, a hyperforest is equivalent to a triangulated graph and hence the treewidth of a triangulated graph is the size of its largest clique minus 1.

Furthermore, the sizes of the minimal separators of the triangulated graph are at most the treewidth.

Treewidth is important in both learning and inference. A Markov network with higher treewidth has greater representative power, encoding more dependencies between variables, and hence can often explain data better. However, the efficiency of most inference algorithms decreases with increasing treewidth, and in learning, a higher treewidth could result in overfitting. Hence it is a common practice to use a Markov network of a bounded treewidth to fit the data.

An alternative definition of treewidth is via tree decomposition which we will not discuss here for brevity [7, 1]. In fact NB makes heavy use of tree decomposition, but we will reformulate their arguments without it.

We now define a k-hypertree. The concept of the hypertree is essential to KS:

Definition 2.9 (Hypertree). *A k-hypertree is a maximal hyperforest of width k, i.e. no more edges can be added to the k-hypertree without increasing its treewidth.*

A more intuitive definition of a k-hypertree might be the following construction:

Definition 2.10 (Hypertree). *A clique with $k+1$ vertices is a k-hypertree. Given a k-hypertree, T_n on n vertices, we can obtain a k-hypertree, T_{n+1} by adding a new vertex to T_n and connecting it to a single k-clique of T_n.*

For example, figure 1(c) shows a 3-hypertree, which can be constructed by beginning with vertices $\{1, 2, 3, 4\}$ (which is a clique of size 4) and adding vertex 5 and connecting it to vertices $\{1, 2, 5\}$, a 3-clique.

The following two theorems explain why triangulated graphs are preferred [7, 5]:

Theorem 2.1 (Hammersley-Clifford). *A random vector X has a probability distribution that factorizes over a Markov network (or hypergraph) with cliques (or hyperedges, H) iff it's probability distribution has the form:*

$$P_X(x) = \prod_{h \in H} \phi_h(x_h) \tag{1}$$

where $\phi_h(x_h)$ is called a clique factor and is a function of the outcomes of random variables of the clique h. Note that the product can be either over all cliques (not necessarily maximal) or over maximal cliques since the non-maximal clique factors can always be absorbed by the maximal clique factors. We say that the probability distribution P *factorizes* over the graph. Note that in general, there does not exist a bijection between a graph and a probability distribution. For example, all probability distributions factorize over the complete graph, and any graph encodes conditional independencies consistent with an unlimited number of probability distributions.

Really, we are interested in calculating the probabilities of random variables in the network, but unfortunately, the clique factors, ϕ might not correspond directly to clique probabilities in general. In particular, a change in marginal probabilities of a particular clique can propagate to clique factors of faraway cliques. Fortunately, in the case of triangulated graphs, clique factors do correspond to clique marginals:

Theorem 2.2 (Clique Factorization in Triangulated Graphs). *Clique factors in triangulated graphs correspond to marginal distributions, i.e.*

$$\phi_h(x_h) = \frac{P_h(x_h)}{\prod_{h' \subset h} \phi_{h'}(x_{h'})} \tag{2}$$

The product is over all cliques, not necessarily maximal.

The following theorem is essential to NB, a partial proof of which is found in [7]:

Theorem 2.3 (Projecting Probability Distributions onto Hyperforests). *Let $P_X(x)$ be a probability distribution, and $H(V)$ a hyperforest. Then the unique minimizer, $\tilde{P}$ of $D(P||\tilde{P})$ (KL-divergence between P and $\tilde{P}$) subject to the constraint that $\tilde{P}$ factorizes over $H(V)$ is*

$$\tilde{P}_X(x) = \frac{\prod_{h \in H^*} P(x_h)}{\prod_{s \in S} P(x_s)} \tag{3}$$

where H^* is the set of maximal hyperedges of $H(V)$ and S is the set of minimal separators of H. In particular,

$$D(P||\tilde{P}) \leq \sum_{s \in S} I_P(V_{s1}; V_{s2}|s) \tag{4}$$

where V_{s1} and V_{s2} are the two disjoint components of H obtained via removing s (we can always group the components into two even if there are more than two disjoint components). $I_P(V_{s1}; V_{s2}|s)$ is the mutual information between V_{s1} and V_{s2} given s, and is zero iff V_{s1} and V_{s2} are independent given s. Mutual information can be evaluated by [2]:

$$I(X; Y|Z) = E_{p(x,y,z)}[log \frac{p(X, Y|Z)}{p(X|Z)p(Y|Z)}] \tag{5}$$

where X, Y, Z are random vectors and $p(\cdot)$ is their probability distribution. $D(\cdot)$ is KL-divergence and is defined as [2]:

$$D(p||q) = \sum_x p(x) log \frac{p(x)}{q(x)} \tag{6}$$

where $p(\cdot)$ and $q(\cdot)$ are both probability distributions of the random vector X, and the sum is over all possible outcomes of X. KL-divergence is a measure of how close q is to p, and is at least zero, and equals to zero

iff $p = q$. Notice the similarity between mutual information and KL-divergence. We can rewrite $I(X;Y|Z)$ as $D(p(X,Y|Z)||p(X|Z)p(Y|Z))$ and thus mutual information measures how close the joint probability distribution of X, Y given Z is to being independent.

Note that if $P_X(x)$ is factorizable over $H(V)$, then equation 3 yields $P_X(x)$ and $D(P||\tilde{P})$ is zero. We can think of equation 4 as summing the error incurred as a result of $H(V)$ assuming conditional independencies when there is none.

3 The Learning Problem

In this section, we define the learning problems that KS and NB are studying.

We would like to estimate an unknown distribution, P_{true} of n variables, based on samples drawn from it. We could use the empirical distribution P_{emp} of the samples but that easily suffers from overfitting and dimensional blow-up due to n. A standard approach would be to limit the class of distribution hypotheses to a smaller subset, T, and attempt to estimate a P^*, such that $P^* = argmin_{P \in T} Q(P|data)$, where $Q(\cdot)$ is some measure of goodness.

In this paper, we will consider the class of distributions, T_k that are factorizable over triangulated graphs, G_k, of treewidth at most k, and we use KL-divergence as our measure of goodness.

Given observed data, and treewidth k, KS aims to find the distribution $P_{ML} \in T_k$ that minimizes $D(P_{emp}||P_{ML})$. It is straightforward to show from the definition of KL-divergence that P_{ML} is also the distribution within T_k that maximizes the likelihood of the observed data. Such a problem is NP-hard, and KS gets around it by first establishing equivalence to the maximum hypertree problem and creating an approximate structure to get a constant factor approximation to the maximum hypertree problem.

On the other hand, given a treewidth k, NB wants to find the distribution $\tilde{P}$ that factorizes over G_k (assuming P_{true} factorizes over at least one graph in G_k), so as to minimize $D(P_{true}||\tilde{P})$. Obviously it is impossible to exactly find the true distribution given any finite number of samples, so as a compromise, they find a polynomial time algorithm that samples the true distribution, P_{true}, a polynomial number of times to find a graph, $\tilde{G} \in G_k$. $\tilde{G}$ has the property that the optimal distribution, $\tilde{P}$ that factorizes over $\tilde{G}$ according to equation 3 has $D(P_{true}||\tilde{P}) < \epsilon$ with probability at least $1 - \delta$. Note that P_{true} might not necessarily be factorizable over $\tilde{G}$. In fact, even in the case when P_{true} does not factorize over G_k, if P_{true} is such that there exists $G \in G_k$, such that $D(P_{true}||\tilde{P}) < \epsilon$, the algorithm can still find the answer with high confidence.

4 Learning to Maximum Hypertree

In this section we describe the equivalence of the learning problem formulated by KS and the maximum hypertree problem.

As mentioned in section 3, we are trying to find $P_{ML} \in T_k$ that minimizes $D(P_{emp}||P_{ML})$. From theorem 2.3 (replacing $P_X(x)$ with $P_{emp}(x)$), we find that for every $G \in G_k$, the best P^* that factorizes over G and minimizes $D(P_{emp}||P^*)$ is given by equation 3. Hence if we go through every triangulated graph (hyperforest) in $G \in G_k$, calculate P^* by equation 3, calculate $D(P_{emp}||P^*)$ and then select P^* with the lowest $D(P_{emp}||P^*)$, we are done. However, there are an exponential number of such graphs and so this is intractable. But let's continue this line of thought: for each hyperforest $G \in G_k$, we want to minimize $D(P_{emp}||P^*)$, where P^* is given by equation 3. Hence,

$$\begin{aligned} min_{G \in G_k} D(P_{emp}||P^*) &= min_{G \in G_k} E_{P_{emp}}[log \frac{P_{emp}}{P^*}] \\ &= C - \sum_{h \in G} w_h \end{aligned} \quad (7)$$

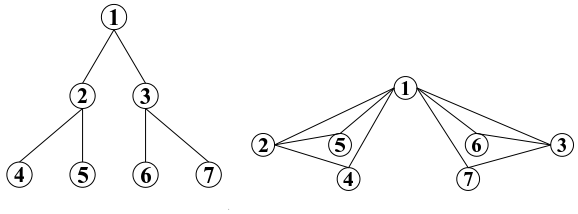
where the sum is over all (not necessarily maximal) hyperedges of G, and C is a constant, same for all hyperforests and

$$w_h = \sum_{h' \subseteq h} (-1)^{|h|-|h'|} E_{P_{emp}}[log(X_{h'})] \quad (8)$$

where we are summing all subsets of h including itself and the expectation term is simply the negative of the entropy of $X_{h'}$. Dropping the constants, this is equivalent to finding a triangulated Markov network (hyperforest) with the largest weight given by $\sum_{h \in G} w_h$. Note that this weight is always negative, but it can be shown that if $G' \subset G$ are both hyperforests, then $\sum_{h \in G} w_h - \sum_{h \in G'} w_h \geq 0$ [7]. This is not at all surprising because $\sum_{h \in G} w_h - \sum_{h \in G'} w_h \geq 0$ simply means that the optimal P^* that factorizes over G is closer to P_{emp} than the optimal P^* that factorizes over G'. But this is obvious since the set of probability distributions that factorize over G is a strict superset of the probability distributions that factorize over G'. We call such a property of the weights *monotonicity*. Formally, we have:

Definition 4.1 (Monotonic Weights). *Given an input k, a weight function w over a set of vertices, V, is a mapping from subsets of V of size $\leq k+1$ to $\mathbb{R}$, i.e. $w : \binom{V}{\leq k+1} \rightarrow \mathbb{R}$. We say $w(\cdot)$ is monotonic if $|h_2| \leq k+1$ and $h_1 \subseteq h_2 \in V$ implies $\sum_{h \subset h_1} w(h) \leq \sum_{h \subset h_2} w(h)$.*

Summarizing this section, this is the problem we would like to solve:



(a) A rooted tree, $T_{r,2}(V)$

(b) 2-windmill based on rooted tree in (a)

Figure 2: Rooted tree and corresponding 2-windmill

Definition 4.2 (The Maximum Hypertree Problem). *Given inputs: treewidth k, vertex set V and monotonic weight function $w : \binom{V}{\leq k+1} \rightarrow \mathbb{R}$, we want to find the heaviest hyperforest $H(V)$, that is the hyperforest that maximizes $\sum_{h \in H} w(h)$. Because of the monotonicity of the weight function, the heaviest hyperforest will be a hypertree, and hence we have the maximum hypertree problem.*

5 Maximum Hypertree to Windmills

Because the maximum hypertree problem is NP-hard, in this section, we explore a type of hypergraph known as the windmill, which will be used to approximate a hypertree. Consider a tree, $T_{r,k}(V)$ over the set of vertices V, with root r and depth at most k.

Definition 5.1 (Level of Vertices). *$\forall v \in T_{r,k}(V)$, the level of v is defined to be the length of the shortest path from r to v. Level(v) is unique because $T_{r,k}(V)$ is a tree.*

For example, in figure 2(a), vertex 1 is at level 0, while vertices 2, 3 are at level 1 and vertices 4, 5, 6, 7 are at level 2.

Definition 5.2 (Windmill). *Consider a hypergraph derived from the rooted tree, $T_{r,k}(V)$, by including all vertices in each path from the root, r to every vertex, v to be a hyperedge. Such a hypergraph is called a k-windmill.*

Hence in figure 2, the sets of vertices $\{1, 2\}$ and $\{1, 2, 4\}$ are both hyperedges because $\{1, 2\}$ and $\{1, 2, 4\}$ specify paths from vertex 1 (the root) to vertex 2 and vertex 4 respectively. The level of a vertex, v in a windmill is equal to the level of the vertex in the rooted tree, $T_{r,k}(V)$ from which the windmill is derived from. Therefore, in the windmill shown, vertex 1 is at level 0, vertices 2, 3 are at level 1 and vertices 4, 5, 6, 7 are at level 2.

Definition 5.3 (Windmill Farm). *A k-windmill farm is a disjoint collection of k-windmills.*

Here are some obvious properties of windmills:

1. k-windmills and k-windmill farms are hyperforests. This can be seen by the Graham reduction beginning with the vertices of level k, $k-1$, $k-2$ and so on. For example, in figure 2(b), we can perform a graham reduction by starting with vertices 4, 5, 6, 7, followed by 2, 3, and finally vertex 1.

2. Since the largest hyperedge in a k-windmill is of size $k+1$, the treewidth of a k-windmill (and hence k-windmill farm) is at most k.

3. 1-windmills are star graphs and 2-windmills look like physical windmills as shown in figure 2.

Theorem 5.1 (Windmill Cover Theorem). *For any hyperforest, $H(V)$ of width k and monotonic weight function $w(\cdot)$, there exists a k-windmill farm $F(V)$ such that $w(H) \leq (k+1)!w(F)$.*

The proof of the windmill cover is a constructive probabilistic proof, so we give a brief outline here. We first prove a version of the theorem for nonnegative weight functions, and then generalize that to monotonic weight functions.

The basic idea behind the proof is that by randomly constructing a k-windmill farm, we can in expectation preserve a $\frac{1}{(k+1)!}$ fraction of the weight, by showing that each hyperedge of size up to $k+1$ is included in the windmill farm with probability $\frac{1}{(k+1)!}$. Then by the definition of expectations there must exist a windmill farm that captures at least that fraction of weight of the maximum hyperforest (i.e. the hypertree).

Consider coloring the vertices of the hypergraph with $k+1$ colors, where we color each vertex of a hyperedge differently. Such a coloring exists: take a leaf v of $H(V)$, recursively color $H(V-v)$ (which is also a hyperforest by the Graham reducibility of hyperforests), and then color v. Since v is a leaf it has at most k neighbors (the hyperedge it is on), so there will definitely be a color remaining for v. This coloring scheme imposes on each hyperedge a color ordering—namely, the order of colors that was used to color that hyperedge. Now consider a random permutation of the $k+1$ colors, π. Our hyperforest F_π is defined to be the set of hyperedges that are *consistent* with the ordering π. A hyperedge is consistent with π if its color ordering forms a prefix or the entirety of π. We need to verify that F_π is indeed a windmill farm, and that in expectation it preserves a $\frac{1}{(k+1)!}$ fraction of the weight of the maximum hypertree. The former is not difficult to intuit—if a vertex of color $i+1$ is included in the windmill farm, then it has to be on a hyperedge with a vertex

of color i, which serves as its parent in the windmill farm. Each hyperedge of size r is selected to be in the windmill farm if it prefixes π, which happens with probability $\frac{(k+1-r)!}{(k+1)!} \geq \frac{1}{(k+1)!}$. Thus the maximum windmill farm should capture at least that fraction of the weight of the maximum hypertree.

To extend this proof to monotonic rather than just nonnegative weight functions, we tweak the analysis slightly. As we color the vertices in reverse Graham reduction order, note that we are incrementally building up the windmill farm one vertex at a time. Each vertex and its associated hyperedge is added only once, and due to monotonicity, the weight of the windmill farm can only increase with each addition. Therefore the same analysis holds for monotonic weight functions.

From the Windmill Cover Theorem, we know that there exists a k-windmill farm that covers at least $\frac{1}{(k+1)!}$ of the weight of the maximum hypertree. Consequently, if we can find the heaviest k-windmill farm over the input vertex set, V, then we have found a $\frac{1}{(k+1)!}$ factor approximation to the maximum hypertree problem.

6 ILP for Windmill Farms

Given inputs treewidth k, vertex set V and monotonic weight function $w : \binom{V}{\leq k+1} \to \mathbb{R}$, we want to find the heaviest k-windmill farm. In this section, we establish an integer linear program whose solution is exactly the heaviest k-windmill farm. Of course, solving an ILP is intractable, but with the appropriate relaxation and randomized rounding scheme, we can obtain a reasonable approximation to the ILP.

First, a brief word on notation. We define a "path" p to be a sequence of vertices from V, and length $|p|$ to be the number of vertices in p. Let $\cdot$ denote the concatenation operator for paths. For example, $p \cdot q$ is path p followed by path q, and $p \cdot v$ is path p followed by vertex v.

Consider associating a variable x_p with each path p of length at most $k+1$. When $x_p = 1$, we consider the vertices of p to be a hyperedge in our k-windmill farm. For example, if $p = (v_1, v_2, ..., v_\alpha)$ for some $\alpha \leq k+1$, then setting $x_p = 1$ implies that $\{v_1, v_2, ..., v_\alpha\}$ is a hyperedge of our k-windmill farm.

In particular, recall from section 5 that a k-windmill farm is induced from a disjoint collection of rooted trees of depth at most k. Setting $x_p = 1$ implies that v_1 or v_α is the root of one of those trees. By convention, we shall assume v_1 to be the root and $(v_1, v_2, ..., v_\alpha)$ is therefore a walk down that tree. However, this also means that there exists a path from root, v_1 to $v_{\alpha-1}$, via $q = (v_1, v_2, ..., v_{\alpha-1})$, where $p = q \cdot v_\alpha$, thus implying that q is also a hyperedge of our windmill farm and $x_q = 1$.

Therefore, we can consider the incremental improvements in the total weight of the windmill farm by adding v_α to our windmill farm, assuming the previous vertices of p are already part of the windmill farm. To be precise, let w_p be the weight of path p. Then we have $w_p = \sum_{h \subseteq p} w(h) - \sum_{h \subseteq q} w(h)$. Because the weight function $w(h)$ is monotonic for h up to size $k+1$, we can conclude that all w_p are nonnegative.

Based on these definitions, we can express our ILP objective function as $\sum_p x_p w_p$, where p ranges over all possible paths from length 0 to $k+1$. Note there are $O(n^{k+1})$ such paths, and thus the number of variables is polynomial.

We also need to ensure that the hypergraph induced from x_p is indeed a k-windmill farm, by enforcing two constraints on our ILP:

1. **Path consistency**: If we decide to include a path $p = (v_1, v_2, ..., v_\alpha)$, we also have to include the path leading up to the second-to-last vertex (defined as q in the above example). In other words, for all p such that $|p| \leq k$, and for all successor vertices v, $x_{p \cdot v} = 1$ only if $x_p = 1$. Expressed as a linear constraint, we have $(\forall p, v) x_{p \cdot v} \leq x_p$ and $x_p \in \{0, 1\}$.

2. **Tree structure**: For the underlying structure of the windmill farm to be a tree, there can only be one path to each vertex from the root of the tree it is in. For all vertices v, there can only be one path q (where $|q| \leq k$) leading to it. Therefore, there exists at most one q where $x_{q \cdot v} = 1$. This can be expressed in constraint form as $(\forall v) \sum_q x_{q \cdot v} \leq 1$ and $x_p \in \{0, 1\}$.

Putting together the objective function and constraints, we have the following ILP:

$$\begin{aligned} max \quad & \sum_{\{p:\ |p| \leq k+1\}} w_p x_p \\ (\forall v) \quad & \sum_{\{q:\ |q| \leq k\}} x_{q \cdot v} \quad \leq \quad 1 \\ (\forall v, |p| \leq k) \quad & x_{p \cdot v} \quad \leq \quad x_p \\ (\forall p) \quad & x_p \quad \in \quad \{0, 1\} \end{aligned} \tag{9}$$

(In our notation we explicitly specify the lengths of the paths, though technically x_p for $|p| > k+1$ is not defined.)

7 LP Relaxation

To make the problem tractable, we now relax the ILP to obtain a linear program. The naive relaxation of the above ILP would be to replace the constraint $x_p \in \{0, 1\}$ with $x_p \in [0, 1]$. However, if we can find a relaxation that has a more restricted feasible polytope, while still containing the feasible solutions of the original ILP, we have a chance

of reducing the integrality gap when we round. With this in mind, consider the following relaxed LP:

$$\begin{aligned} & max \sum_{\{p:\ |p|\leq k+1\}} w_p x_p \\ (\forall v, |p| \leq k) \sum_{\{q:\ |q|\leq k-|p|\}} x_{p\cdot q\cdot v} & \leq x_p \\ (\forall p) \quad x_p & \geq 0 \\ x_\epsilon & = 1 \end{aligned} \tag{10}$$

First, let us verify that this relaxation sufficiently captures the original ILP constraints. Note that when considering just the $q = \epsilon$ term, the constraint becomes $x_{p\cdot v} \leq x_p$, the path consistency constraint from the ILP. Additionally, when $p = \epsilon$, the constraint becomes $\sum_q x_{q\cdot v} \leq 1$, the tree constraint from the ILP.

We also need to verify that the original ILP feasible polytope is contained within this LP's feasible solution set. When we restrict our consideration to integral x_p values, note that the $p = \epsilon$ case specifies that there is only one $x_{p\cdot v}$ for each v that can be one. Therefore, given path consistency, we can conclude the LP constraint above.

8 Iterative Randomized Rounding

The relaxed LP will generally give fractional solutions, which by themselves do not specify a proper windmill farm structure, and thus cannot be used as the maximum hypertree. We need to round the LP solutions to integral values, while still satisfying the original feasibility constraints of the ILP. A straightforward one-pass rounding (where we round each x_p to one with probability x_p) is unlikely to satisfy the path consistency and tree structure constraints of the ILP, because we could very well round some x_p to zero but $x_{p\cdot v}$ to one, or both $x_{p\cdot v}$ and $x_{q\cdot v}$ to one (where p and q are different paths). Thus, we'd like to round in a way that preserves both path consistency and tree structure.

To do so, we round in an *iterative* fashion. At iteration i, for i from 1 to k, we round all variables x_p where $|p| = i$. To guarantee path consistency, we automatically round $x_{p\cdot v}$ to zero if x_p is zero. To guarantee tree structure, for any given vertex v we only round at most one of the variables $x_{p\cdot v}$ to one. In particular, if some $x_{p\cdot v}$ was already rounded to one in an earlier iteration (i.e., for a shorter path), we are not allowed to round any future vertex ending in v to one. Between every iteration of rounding, we should readjust the not-yet-rounded x_p values so that they continue to be optimal. We do so by re-solving the LP at each iteration, using the previously rounded variables as constraints.

Let LP^i denote the ith iteration's linear program, where LP^0 is the original relaxed LP defined in the previous section. Let x^i be the respective optimal solution for LP^i. Let $\tilde{x}$ represent the rounded variables. Putting these ideas together, we obtain the following rounding scheme:

1. Solve LP^0 to obtain x^0.
2. For i from 1 to k:
   (a) For each vertex v:
      i. Randomly select a path p with probability $x^{i-1}_{p\cdot v}$ where $|p| = i-1$, or no path at all with probability $1 - \sum_{\{p:\ |p|=i-1\}} x^{i-1}_{p\cdot v}$
      ii. If a path p is selected, set $\tilde{x}_{p\cdot v}$ to one
   (b) Set all remaining $\tilde{x}_p$ where $|p| = i$ to zero
   (c) Add newly rounded variables $\tilde{x}$ as constraints to LP^{i-1} to obtain LP^i
   (d) Solve LP^i to obtain x^i
3. Return x^k (which will be integral)

This scheme rounds variables associated with successively longer paths. We guarantee path consistency—when we round some x_p to zero, we add $x_p = 0$ to all future LPs, thereby enforcing that for any q, $x_{p\cdot q}$ is constrained to be zero. We also guarantee tree structure—when we round some $x_{p\cdot v}$ to one, we cannot round $x_{q\cdot v}$ to one. To see why, consider two cases. If $|q| = |p|$, we can only select either one of them to round to one, because they are selected within the same iteration. If $|q| > |p|$, the LP includes a constraint for tree structure that prevents $x_{q\cdot v}$ from taking on any value but zero.

It remains to show that this rounding scheme introduces a reasonably small bounded integrality gap.

Theorem 8.1 (Integrality Gap). *The optimal value for LP^i is within a $\frac{1}{8(k+1-i)}$ factor of the optimal value for LP^{i-1}.*

Due to space constraints, here we give an intuitive argument why this theorem holds. Rather than showing that the *optimal* solution to LP^i is within a $\frac{1}{8(k+1-i)}$ factor of LP^{i-1}'s optimal, we show that there exists a *feasible* solution to LP^i that is within that factor. The optimal LP^i solution (that is, x^i) can only have as good or better of a value. Consider this solution $x^{(i)}$ to LP^i:

- For variables already rounded (that is, $\tilde{x}_p$ for $|p| < i$), LP^i constrains that $x^{(i)}_p = \tilde{x}_p$.
- For variables $x_{p\cdot q}$ where $|p| = i$ and $\tilde{x}_p = 0$, set $x^{(i)}_{p\cdot q} = 0$ to ensure path consistency.
- For variables $x_{p\cdot q}$ where $|p| = i$ and $\tilde{x}_p = 1$, set $x^{(i)}_{p\cdot q}$ to

$$\frac{x^{i-1}_{p\cdot q}}{4(k+1-i)x^{i-1}_p} \tag{11}$$

- For each vertex v, if the above settings cause $\sum_p x^{(i)}_{p\cdot v}$ to exceed one, we set any variable which has a v on its path to zero. That is, any variable $x^{(i)}_{p\cdot v\cdot q}$ (for all p and q) is set to zero, causing the sum to go to zero. In this situation, vertex v *overflowed* so we *purge* all paths that include v.

We need to show that $x^{(i)}$ is a feasible solution, and that in expectation it preserves $\frac{1}{8(k+1-i)}$ fraction of the optimal value of LP^{i-1}.

First, we argue that this solution is feasible. We need to verify that $\sum_{\{q:|q|\leq k-|p|\}} x^{(i)}_{p\cdot q\cdot v} \leq x^{(i)}_p$ is satisfied for all p. Consider these cases for the constraint:

- $|p| = 0$: The purging step guarantees $\sum_q x^{(i)}_{q\cdot v} \leq 1$, which is the $p = \epsilon$ constraint.
- $1 \leq |p| \leq i$, $\tilde{x}_p = 0$: This constraint already appeared in LP^{i-1}, and the entire sum was set to zero, and remains zero.
- $1 \leq |p| \leq i$, $\tilde{x}_p = 1$: The sum over paths starting with p is obviously less than or equal to the sum over all paths, which is the $|p| = 0$ case. Thus, the sum is at most 1, which is x_p.
- $|p| > i$: Write p as $r \cdot s$ where $|r| = i$. Now consider two subcases:
  - $\tilde{x}_r = 0$ or a vertex of p overflows: The sum is zeroed in either case, so the constraint is satisfied.
  - $\tilde{x}_r = 1$ and no vertex of p overflows: Ignoring purging, the constraint is equivalent to a constraint of LP^{i-1}, except scaled down by a factor of $\frac{1}{4(k+1-i)x_p^{i-1}}$, so it is satisfied. Note that purging can only make the constraint more strongly satisfied.

Second, we argue that we preserve a $\frac{1}{8(k+1-i)}$ fraction of the optimal value of LP^{i-1}. Ignoring purging for the moment, consider a path $p\cdot q$ where $|p| = i$. x_p is rounded to one with probability x_p^{i-1}, in which case path $x_{p\cdot q}$ will contribute a $\frac{1}{4(k+1-i)x_p^{i-1}}$ fraction of the weight $w_{p\cdot q}$ that $x^{i-1}_{p\cdot q}$ did. Thus, *in expectation*, path $p \cdot q$ will contribute a $\frac{1}{4(k+1-i)}$ fraction of the weight after rounding. However, if the path is purged, then it will contribute no weight at all. Therefore we'd like to bound the probability that the path is purged, i.e., that any vertex v on $p \cdot q$ overflows. By the LP^{i-1} constraints, the incoming value ($\sum_p x_{p\cdot v}$) to a vertex v via paths that share the prefix p is at most $\frac{1}{4(k+1-i)}$. For paths that do not share the prefix p, the expected contribution is even less. Adding these two together, the total incoming value to v is at most $\frac{1}{2(k+1-i)}$. By summing over all $k+1-i$ vertices of q, in expectation there is $1/2$ weight entering the path q. By the Markov inequality, this implies that with at least $1/2$ probability we will not purge the path. Thus, the overall expected value for $x_{p\cdot q}$ is $\frac{1}{8(k+1-i)}$. By linearity of expectation, and because the LP objective function is linear, this implies that in expectation $x^{(i)}$ preserves a $\frac{1}{8(k+1-i)}$ fraction of the weight of the previous LP optimum x^{i-1}.

Applying theorem 8.1 for $i = 1$ to k readily gives us this overall integrality gap:

Corollary 8.2 (Overall Integrality Gap). *The rounding scheme returns an integral solution that is within a $\frac{1}{8^k k!}$ factor of the optimal value for LP^0.*

Thus, compounded with the windmill approximation gap of $\frac{1}{(k+1)!}$, our approximation to maximum hypertree is in expectation within a factor of $\frac{1}{8^k k!(k+1)!}$.

We have demonstrated a constant factor approximation for maximum hypertree, as we set out to do. This concludes our study of KS's work.

9 Minimizing $D(P_{true}||\tilde{P})$

We recall from section 3 that NB [5] presents a polynomial time algorithm that samples the true distribution, P_{true} a polynomial number of times to find a graph, $\tilde{G} \in G_k$. $\tilde{G}$ has the property that the optimal distribution, $\tilde{P}$, that factorizes over $\tilde{G}$ as given by equation 3 has $D(P_{true}||\tilde{P}) < \epsilon$ with probability at least $1 - \delta$.

The basic strategy comes from theorem 2.3, equation 4: the projection error of P_{true} on a given triangulated graph (hyperforest), G is $\leq \sum_{s\in S} I_P(V_{s1}; V_{s2}|s)$, where S is the set of minimal separators of G. Because there are at most $|V| = n$ terms (equal to number of random variables) in the summation, if we can ensure that each term is small enough, we can be confident of the error being below ϵ.

Because we only consider triangulated graphs G_k of at most treewidth k, the minimal separators are at most of size k. Hence there are only $O(n^k)$ (polynomial) number of separators to consider. If for each possible separator s we can divide the remaining vertices (variables) into two groups V_{s1} and V_{s2} in such a way as to approximately minimize or upper bound $I_P(V_{s1}; V_{s2}|s)$, then our task is almost complete. To do this, NB [5] makes use of an algorithm from another paper [6], as a black-box subprocedure of their algorithm.

The problem is complicated by the fact that we do not know P_{true} although theorem 2.3 depends on us knowing P_{true}. To get around this, we observe that we are only interested in minimizing $\sum_{s\in S} I_P(V_{s1}; V_{s2}|s)$. For certain classes of distributions, we can estimate entropies (and

hence mutual information) of arbitrary subsets of random variables with precision ϵ and confidence $1-\delta$ using polynomial (in n, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$) number of samples. For example, the generalized Chernoff-Hoeffdings bound shows this is possible for discrete distributions, and specialized techniques exist for certain continuous distributions such as Gaussians.

10 Submodularity

We first define the concept of submodular and symmetric functions:

Definition 10.1 (Submodularity). *A function $f : 2^V \rightarrow \mathbb{R}^+$ is submodular if $\forall A, B \subseteq V$, $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$.*

Definition 10.2 (Symmetric). *A function $f : 2^V \rightarrow \mathbb{R}^+$ is symmetric if $f(A) = f(V \backslash A)$.*

Let $S \subset V$ and let $V_S = V \backslash S$. A useful submodular and symmetric function is mutual information: $F_{P,S} : 2^{V_s} \rightarrow \mathbb{R}^+$,

$$\begin{aligned} F_{P,S}(A) &= I_P(A; V_S \backslash A | S) \\ &= E_P[log \frac{P(A, V_S \backslash A|S)}{P(A|S)P(V_S \backslash A|S)}] \end{aligned} \tag{12}$$

From the definition of mutual information, $F_{P,S}$ is clearly symmetric. The proof of submodularity is easily done by plugging in the definition of mutual information.

Given a symmetric and submodular function, $f : 2^V \rightarrow \mathbb{R}^+$ using Queyranne's Algorithm [6] (henceforth known as QA), we can obtain a proper subset $A^* \subset V$ such that $A^* = argmin_{B \in 2^V \backslash \{V, \phi\}} f(B)$ using only $O(|V_s|^3)$ calls to evaluate f. Suppose we input the function $F_{P,S}$ into QA, where $S \subseteq V$, and find that $F_{P,S}(A^*) \leq \epsilon$. Then we say that S is an ϵ-separator of V and $(A, V \backslash A)$ is an ϵ-partition of (V_S, S, P).

In our case however, we do not know P_{true}, but, we shall assume the existence of a sampler so that we can estimate $F_{P,S}$ in polynomial time. In particular, we shall assume that the sampler gives an estimate $F_{\tilde{P},S}$ upon query, such that $\forall A \subseteq V_S$

$$|F_{\tilde{P},S}(A) - F_{P,S}(A)| \leq \epsilon_1 \text{ w.p. } \geq 1 - \delta_1 \tag{13}$$

Now, being random, $F_{\tilde{P},S}$ might no longer be submodular, since there is no guarantee that separate queries of $F_{\tilde{P},S}(A)$, $F_{\tilde{P},S}(B)$, $F_{\tilde{P},S}(A \cup B)$ and $F_{\tilde{P},S}(A \cap B)$ will yield consistent estimates such that $F_{\tilde{P},S}(A) + F_{\tilde{P},S}(B)$ is at least $F_{\tilde{P},S}(A \cup B) + F_{\tilde{P},S}(A \cap B)$. Because of this, $\tilde{A}$, the result returned by QA might no longer be the minimizer of $F_{\tilde{P},S}$.

However, this submodularity gap can be bounded:

Theorem 10.1 (Submodularity Gap). *Suppose $F_{P,S} : 2^V \rightarrow \mathbb{R}^+$ is a symmetric submodular function and $F_{\tilde{P},S} : 2^V \rightarrow \mathbb{R}^+$ is another function that is not necessarily submodular, but satisfies $|F_{\tilde{P},S}(A) - F_{P,S}(A)| \leq \epsilon_1$ $\forall A \subseteq V_S$, then QA will return a non-empty proper subset $\tilde{A} \subset V_S$, such that for all non-empty proper subset $A \subset V_S$,*

$$F_{\tilde{P},S}(\tilde{A}) - F_{\tilde{P},S}(A) \leq |V| \cdot \epsilon_1 \tag{14}$$

Because our sampler only guarantees ϵ_1 precision with probability $1 - \delta_1$, theorem 10.1 holds with probability $1 - |V_S|^3\delta_1$ using the union bound since QA will only make $O(|V_s|^3)$ queries to the sampler.

Therefore $\tilde{A}$ is an approximate minimizer of $F_{\tilde{P},S}$ and can also be easily shown to be the approximate minimizer of $F_{P,S}$: $\forall A, F_{P,S}(\tilde{A}) - F_{P,S}(A) \leq (|V| + 2)\epsilon_1$. We can intuitively see this: $|V|\epsilon_1$ comes from the modularity gap (theorem 10.1), while $2\epsilon_1$ comes from possibly underestimating $F_{P,S}(\tilde{A})$ and overestimating $F_{P,S}(A)$ (see equation 13). Once again this is true with probability $1 - |V_S|^3\delta_1$. Summarizing, we have

Theorem 10.2 (Approximation of $F_{P,S}$). *Suppose $F_{P,S} : 2^V \rightarrow \mathbb{R}^+$ is a symmetric submodular function and $F_{\tilde{P},S} : 2^V \rightarrow \mathbb{R}^+$ is another function that is not necessarily submodular, but satisfies $|F_{\tilde{P},S}(A) - F_{P,S}(A)| \leq \epsilon_1$ $\forall A \subseteq V_S$ with probability at least $1 - \delta_1$, then running QA with $F_{\tilde{P},S}$ returns a $\tilde{A}$, such that for all A, with probability at least $1 - |V_S|^3\delta_1$,*

$$F_{P,S}(\tilde{A}) - F_{P,S}(A) \leq (|V| + 2)\epsilon_1 \tag{15}$$

Suppose $\{A, B\}$ is a 0-partition of (V_S, S, P) (that is $I_P(A; B|S) = 0$), then with probability at least $1 - |V_S|^3\delta_1$, $\{A, B\}$ will also be an ϵ_1-partition of $(V_S, S, \tilde{P})$. On the other hand, if {A,B} is an ϵ_1-partition of $(V_S, S, \tilde{P})$, then with probability at least $1 - |V_S|^3\delta_1$, $\{A, B\}$ will be a $2\epsilon_1$-partition of (V_S, S, P).

11 Partitions

It may happen that disconnecting a separator S from a graph might result in more than two disjoint components. We can always union the components so that we end up with two components only, but if we decide not to, let $\pi = \{A_1, A_2, \cdots, A_m\}$ be the set of disjoint components. Extending our old terminologies, we call π an ϵ-partition of (V_S, S, P) if $I_P(A_i; A_j|S) \leq \epsilon \; \forall i \neq j$.

Using this terminology, we have the following theorem:

Theorem 11.1 (Multiple Components).

1. *If $A \subseteq V_S$ and π is an ϵ-partition for (V_S, S, P), then $\pi^A = \{A_i \cap A : A_i \in \pi\}$ is an ϵ-partition for (A_S, S, P) because $I_P(A_i \cap A; A_j \cap A|S) \leq I_P(A_i; A_j|S) \leq \epsilon$.*

2. *Conversely, suppose π is an ϵ_2-partition for (V_S, S, P), and $A \in \pi$. If ϕ is an ϵ_3-partition for (A_S, S, P), then $\psi = (\pi \backslash A) \cup \phi$ is a $max(\epsilon_2, \epsilon_3)$-partition for (V_S, S, P)*

This theorem is intuitive. Part 1 says that if you start with an ϵ-partition, say $\{(a,b),(c,d,e),(f,g,h,i)\}$, and form a new set by drawing elements from each component, say $\{(a,b) \cup (c,d) \cup (f)\}$, then $\{(a,b),(c,d),(f)\}$ is still an ϵ-partition of the new set. This is not surprising because reducing the number of elements in each partition component won't provide partition components with more information about each other than before.

Part 2 says that if you start with an ϵ_2-partition, say $\{(a,b),(c,d,e),(f,g,h,i)\}$, and split up one of the partition component into an ϵ_3-partition, say $\{(f,g),(h),(i)\}$, then the combined partition $\{(a,b),(c,d,e),(f,g),(h),(i)\}$ is now a $max(\epsilon_2, \epsilon_3)$-partition for (V_S, S, P). Once again this is not surprising since splitting $\{(f,g,h,i)\}$ up won't give the original partition components more information about each other. The only thing that might happen is that $\epsilon_3 > \epsilon_2$, which means that $\{(f,g),(h),(i)\}$ have more information about each other than about the original components.

Note that the above theorem still holds when replacing P with $\tilde{P}$. Using the above theorem, given a set $S \subseteq V$ and $\epsilon > 0$, we can find a ϵ-partition, $\pi(\epsilon)$ for $(V_S, S, \tilde{P})$. Because we are using QA with $F_{\tilde{P},S}$, the ϵ-partition we found might not be the "best" one, but it is still an ϵ-partition for $(V_S, S, \tilde{P})$:

1. $\pi_S^0 \leftarrow \{V_S\}$; $i \leftarrow 0$;
2. while $\exists X^i \in \pi_S^i$ such that $\{A_i, B_i\}$ is an ϵ-partition of $(X_S^i, S, \tilde{P})$
   (a) $\pi_S^{i+1} \leftarrow (\pi_S^i \backslash X^i) \cup \{A_i, B_i\}$
   (b) $i \leftarrow i + 1$
3. $\pi_S(\epsilon) \leftarrow \pi_S^i$

Note however, that the algorithm is run using $F_{\tilde{P},S}$, and so we obtain an ϵ-partition for $(V_S, S, \tilde{P})$, but what we are really interested in, is how this ϵ-partition for $(V_S, S, \tilde{P})$ relates to (V_S, S, P). We need another definition before stating the relationship:

Definition 11.2 (Refinement). *A partition ψ is a refinement of π if every element of π can be written as union of elements of ψ.*

So for example, $\{(a,b),(c,d,e),(f,g),(h),(i)\}$ is a refinement of $\{(a,b),(c,d,e),(f,g,h,i)\}$

Invoking theorems from section 10 and combining the results from this section, we get:

Theorem 11.3 (CIP). *Let CIP (Conditional Independence Partitions) be a collection of partitions $\{(S, \pi_S(\epsilon_2 + (|V| + 2)\epsilon_1) : |S| \geq k\}$ found with the algorithm just stated. Then QA will make at most $O(|V|^{k+5})$ queries to $F_{\tilde{P},S}$, and so with probability at least $1 - |V|^{k+5}\delta_1$:*

1. *$\forall (S, \pi_S) \in CIP$, π_S refines any ϵ_2 partition of (V_S, S, P).*
2. *Conversely, if you take any $(S, \pi_S) \in CIP$, and union π_S's components into a partition $\{A, B\}$ such that π_S refines it, then $\{A, B\}$ is an $|V|^3(\epsilon_2 + 3\epsilon_1)$-partition for (V_S, S, P).*

12 Partitions to Graphical Model

We first need to define compatibility:

Definition 12.1 (Compatibility). *We say that a particular triangulated Markov network, G is compatible with CIP if for every $s \in S$ (the set of minimal separators of G) with corresponding partition, $\{V_s^A, V_s^B\}$ there exists $(s, \pi_s) \in CIP$ which refines it.*

For example, in figure 1(b), $\{5\}$ is a minimal separator whose corresponding partition is $\{(1,2,3,4),(6,7)\}$, if CIP contains $(\{5\},\{(1,2,3,4),(6,7)\})$ or $(\{5\},\{(1,2),(3,4),(6,7)\})$ and so on, we are satisfied. We then have to check the other minimal separators of the graph of interest.

Putting everything together, we remind the readers of theorem 2.3 and how it's key to the problem NB is solving. From theorem 2.3, the projection error of projecting P_{true} onto a graphical model, $G \in G_k$, is given by $\sum_{s \in S} I_P(V_{s1}; V_{s2}|s)$, where S is the set of minimal separators of G. Suppose G is compatible with CIP, we can then make the following observations:

1. For a given graphical model, there cannot be more minimal separators than there are vertices, hence $|S| < |V|$. The maximum number of minimal separators is actually $|V| - 2$ which occurs when there's a Markov chain of vertices.
2. Furthermore, since G is compatible with CIP, this means that $\forall s \in S$ with corresponding partition $\{V_s^A, V_s^B\}$, $\exists\ (s, \pi_s) \in CIP$ which refines it, and this implies by theorem 11.3, that $I_P(V_s^A, V_s^B|s) \leq |V|^3(\epsilon_2 + 3\epsilon_1)$.
3. Combining the first two points, it means that if we can find a G that is compatible with CIP, we have achieved a projection error (KL-divergence) $\leq |V|^4(\epsilon_2 + 3\epsilon_1)$ (with probability at least $1 - |V|^{k+5}\delta_1$). Thus by setting $\delta_1, \epsilon_1, \epsilon_2$ appropriately,

we can ensure an error $\leq \epsilon$ with probability at least $1 - \delta$.

Finding the graph that is compatible with CIP in polynomial time involves a deterministic dynamic programming algorithm. Essentially, we consider increasingly large sets of vertices, and check if they form a compatible graph. A vertex set has a compatible graph if it can be split into two components, both of which have compatible graphs. The algorithm terminates when it finds a partitioning where each component of the partition has a compatible graph, and thus the final graph can be recursively built using the memoized subgraphs. We omit the grungy details here (as the algorithm and proof do not rely on randomization or randomized techniques) and refer the reader to [5] for the full exposition.

13 Conclusion

In this paper we studied the problem of learning the structure of a Markov network. We studied two papers that considered the same restricted class of such networks, bounded treewidth models. These models allow us to restrict the number of dependencies between vertices, to allow for fast operations and less overfitting. Karger and Srebro's work focused on finding the "best" such model given some arbitrary observation data, and demonstrated an algorithm that is able to come to within a constant factor of the best model. Narasimhan and Bilmes looked at a related problem of "recovering" a bounded treewidth model from observation data that is assumed to have been generated from such a model. They proved that if we are allowed to sample a polynomial number of times from the true model, we could construct an arbitrarily good approximation to the true model with high confidence. NB's analysis applies only when the observations are actual samples from a bounded treewidth distribution; if the true distribution is not of that type, the algorithm may not return an answer. KS does not assume control over the way the observations were sampled, and will find the best bounded treewidth model regardless of the true underlying distribution of the data. However, it only makes a guarantee of the "goodness" of the resulting model with respect to the maximum likelihood bounded treewidth model—whether this maximum likelihood model is close to the true underlying distribution, or even the empirical distribution induced by the observations, is uncertain.

There is an abundance of future work possible for both papers. For KS, the current approximation factor, though constant with respect to n, is very explosive with respect to k. There are multiple conceivable ways to try to reduce this factor. First, we could try to tighten the current analyses for both windmill cover and integrality gap. Indeed, work by Liang and Srebro has tried to empirically close the windmill cover gap [4]. Second, we could try to find a structure that captures more of the weight of the hyperforest than windmill farms currently do, while still having a reasonable integrality gap when it comes to the relaxed linear program. One of our original project ideas was along this direction—we studied what we called a (k, d)-*hydra swarm*[1], a generalization of windmill farms that allows each constituent tree to have depth d, and hyperedges (compared to a windmill farm) are taken to be paths of length $k + 1$ in the hydra. Presumably a (k, d)-hydra swarm for $d > k$ should cover a larger portion of the hypergraph at the cost of a larger integrality gap. (For example, a $(1, n)$-hydra swarm can cover the entirety of a graph of treewidth one—it is simply the tree itself.) Third, we could forego the intermediary structure entirely and try to devise an approximation to the maximum hyperforest directly, such as by an ILP. We (unsuccessfully) tried this route as well, though the resulting ILP formulations we found were invariably too large and complex.

For NB, the most obvious future work would be to generalize their techniques to classes of Markov networks beyond bounded treewidth models. As we did not explore the NB paper until much later in our project cycle, we have not yet considered this problem.

References

[1] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, pages 1–21, 1993.

[2] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[3] David Karger and Nathan Srebro. Learning markov networks: Maximum bounded tree-width graphs. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, pages 392–401, 2001.

[4] Percy Liang and Nathan Srebro. How much of a hypertree can be captured by windmills? Technical report, Massachusetts Institute of Technology, 2003. Available at http://people.csail.mit.edu/nati/HyperTrees/.

[5] Mukund Narasimhan and Jeff Bilmes. Pac-learning bounded treewidth graphical models. In *Proceedings*

[1]The Lernaean Hydra of Greek mythology was a fearsome many-headed creature defeated by Heracles. Arguably, the many strongly connected paths emanating from the root of this structure resemble the long necks and heads of the hydra, though at the present we have not successfully emulated Heracles in defeating the hydra and unlocking its hydra cover and integrality gap secrets. However, Heracles did not have to battle an entire *swarm* of hydras, unlike the valiant Chen and Yeo, who both in the process sustained severe injuries, especially to their research organs.

of the 20th conference on Uncertainty in Artificial Intelligence, pages 410–417, 2004.

[6] M. Queyranne. Minimizing symmetric submodular functions. *Math. Programming*, pages 3–12, 1998.

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

[8] Jonathon S. Yedidia, William T. Freeman, and Yair Weiss. Understanding beilef propogation and its generalizations. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2001.