Maximum Likelihood Bounded Tree-Width Markov Networks

Nathan Srebro, Massachusetts Institute of Technology (*nati@mit.edu*)

Density Estimation with Tractable Models

Given T observations of n variables $X_1..X_n$, we would like to estimate the distribution from which they were sampled, using a computationally tractable model. The model can then be used for inference and other calculations on the distribution.

Note: the goal is to approximate the source distribution, and success is measured by divergence from this distribution (not by how well the structure of the model captures the structure of the distribution).

We focus on maximum likelihood density estimation: the maximum likelihood distribution from among some limited class of distributions is sought. Restricting to a limited class of distributions serves both to ensure tractability, and as a statistical regularization.

Chow and Liu (1968) studied the problem of finding the maximum likelihood Markov tree and presented an efficient, and exact, solution, by casting the problem as the combinatorial problem of finding a maximum weight tree. We extend this approach to more complex classes of (undirected) Markov networks.

Bounding the Complexity

•Small clique size: effectively bounds the number of parameters in the model.

•Since explicit calculation are only possible on triangulated graphs, a non-triangulated Markov network is usually triangulated.

•**Tree-width of a graph:** minimum over all triangulations, of the maximum clique size of the triangulation, minus one.

Bounding the tree-width is a good constraint for ensuring tractability, but is not the best constraint for statistical regularization

Problem Statement:

Maximum Likelihood Narrow Markov Networks

• For a specified k, maximum likelihood Markov network of tree-width at most k.

• Equivalently, maximum likelihood Markov network over a triangulated graph with cliques of size at most k+1.

Special case of k=1 is maximum likelihood trees (Chow and Liu).

Heuristic local-search suggested by Malvestuto, 1991.

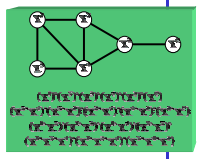
Decomposing the Maximum Likelihood

We would like to decompose the maximum likelihood of a triangulated Markov network to a sum of contributions from "local components". For trees (k=1) the ML can be decomposed to the sum of edge contributions. For higher tree-width edge-contributions are not enough, and larger cliques need to be considered.

In order to decompose the maximum likelihood, we must first study the following explicit factorization of a Markov network over a triangulated graph:

$$P_X(x) = \prod_{\text{Cliques } h} \varphi_h(x_h)$$

$$\varphi_h(x_h) = \frac{P(x_h)}{\prod_{h' \subset h} \varphi_{h'}(x_{h'})}$$



Note that the product is taken over *all* complete subgraphs, not only maximal cliques (we refer to any complete subgraph as a clique). Why not subsume smaller cliques in maximal cliques?

Under this factorization, a clique's factor depends ***only*** on the marginal distribution inside the clique. It does ***not*** depend on the graph structure.

Applying this factorization to the maximum likelihood distribution (which agrees with the empirical distribution on the clique marginals), we can decompose the maximum likelihood of a triangulated graph:

$$\begin{aligned}
\log ML(G) &= \log \prod_t \prod_{h \in Clique(G)} \hat{\varphi}_h(x_h^t) \\
&= \sum_{h \in Clique(G)} \sum_t \log \hat{\varphi}_h(x_h^t) \\
&= \sum_{h \in Clique(G)} \sum_{x_h} T \cdot \hat{P}_h(x_h) \log \hat{\varphi}_h(x_h) \\
&= T \sum_{h \in Clique(G)} \mathrm{E}_{\hat{P}}[\log \hat{\varphi}_h(\mathrm{X}_h)] \\
&= \sum_{h \in \mathrm{Cliques}(G)} w(h) \\
&= \log ML(\phi) + \sum_{h \in \mathrm{Cliques}(G), |h|>1} w(h)
\end{aligned}$$

Where the weights $w(h)$ are given by:

$$\begin{aligned}
w(h) &= \mathrm{E}_{\hat{P}}[\hat{\varphi}_h(\mathrm{X}_h)] \\
&= \mathrm{E}_{\hat{P}}\left[\frac{\hat{P}(x_h)}{\prod_{h' \subset h} \hat{\varphi}_{h'}(x_{h'})}\right] \\
&= \mathrm{E}_{\hat{P}}[\log \hat{P}(x_h)] - \sum_{h' \subset h} \mathrm{E}_{\hat{P}}[\log_{\hat{\varphi}_{h'}}(\mathrm{X}_{h'})] \\
&= -H(\hat{P}(h)) - \sum_{h' \subset h} w(h')
\end{aligned}$$

$$w(h) = -\sum_{h' \subseteq h} (-1)^{|h|-|h'|} H(\hat{P}(h'))$$

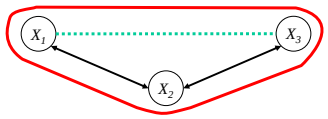
Weight of a Doubleton

$$\begin{aligned}
w(1,2) &= -H(\hat{P}_{\{1,2\}}) - w(1) - w(2) \\
&= -H(\hat{P}_{\{1,2\}}) + H(\hat{P}_1) + H(\hat{P}_2) \\
&= I_{\hat{P}}(1;2) \geq 0
\end{aligned}$$

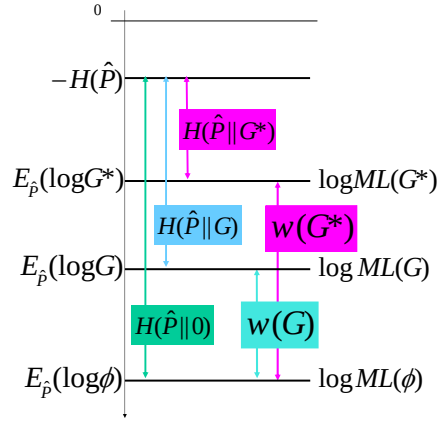
Weight of a Tripleton with no Pairwise Interactions

$$\begin{aligned}
w(1,2,3) &= H(1) + H(2) + H(3) - H(1,2,3) \\
&= D(\hat{P}_{\{1,2,3\}} \| \hat{P}_1 \cdot \hat{P}_2 \cdot \hat{P}_3) \geq 0
\end{aligned}$$

Weight of a Markov Chain as a 3-Clique



$$\begin{aligned}
w(1,2,3) &= H(1,3) - H(1) - H(3) \\
&\quad + H(1,2) + H(2,3) - H(2) - H(1,2,3) \\
&= -I(1;3) < 0
\end{aligned}$$



$$\log ML(G) = \log ML(\phi) + \sum_{h \in \mathrm{Cliques}(G), |h|>1} w(h)$$

The Maximum-Weight k-Hypertree Problem

Given:

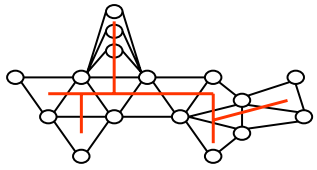
–a width k,

–and a weight on each candidate clique of size at most k+1

Find a triangulated graph with clique size at most k+1 that maximizes the sum of weights of its cliques.

What are Hypertrees ?

The clique-set of a triangulated graph forms an acyclic hypergraph (or *hyperforest).* A hypertree is a maximal acyclic hypergraph of bounded width (hyperedge-size). This is a 2-hypertree:



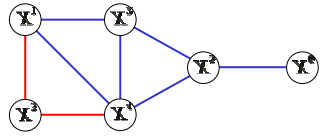
An acyclic hypergraph can also be defined in terms of its tree decomposition, which corresponds to a **junction tree** of a triangulated graph. Other equivalent definitions can be found in the literature.

Bounded tree width graphs (or equivalently narrow acyclic hypergraphs) are important in many applications. There is much work on finding narrow triangulations (super-graph problems) and on using the low tree-width. But there is not much work on finding narrow sub-graphs, which is the problem we are interested in.

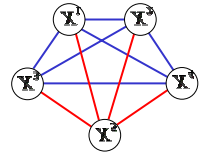
Unlike the maximum weight tree problem, which can be solved in essentially linear time ($O(|E|+|V|\log|V|)$), the maximum hyper-tree problem is hard. It is hard even for k=2, and when weights are given only for 2-edges (and not 3-edges) and all weights are zero/one.

Monotone Weights

The total summed weight of a triangulated graph is no less than the total summed weight of a contained triangulated subgraph:



If the weights are monotone on cliques, they are also monotone on arbitrary triangulated graph. Accordingly, it is enough to require that adding a single vertex to a clique does not decrease the total summed weight:



Hardness

Finding a maximum hypertree is NP-hard. But showing how the maximum likelihood narrow Markov network can be found by finding a maximum hypertree is not enough in order to demonstrate the hardness of the maximum likelihood problem. What is needed is to show how to find the maximum hypertree by solving the maximum likelihood problem.

We will show how a restricted form of the maximum hypertree problem, where weights are specified only for cliques of size exactly *(k+1)*, can be reduced to the maximum likelihood narrow Markov network problem. We will do this by constructing, for any non-negative weight function on *(k+1)* cliques, an appropriate empirical distribution which yields weights proportional to the desired weights.

Creating a Distribution that Yields Desired Weights

•Input: Non-negative weights on *(k+1)* cliques.

•We construct a distribution P over n binary variables. P will have uniform marginals on all subsets of at most k variables, and specifically chosen biases of on the parity of sets of $k+1$ variables.

•P will be a uniform mixture of $\binom{n}{k+1}$ distributions P_h, one for each candidate clique h of size $k+1$.

•Each P_h will be almost uniform, except for a bias of b_h on the parity of the variables in h:

$$P_h(x) = \begin{cases} (1+b)2^{-n} & \text{if } parity(x_h) = 0 \\ (1-b)2^{-n} & \text{if } parity(x_h) = 1 \end{cases}$$

•Since the mixture P is uniform on all marginals of at most k variables, it will have zero weight on the corresponding candidate cliques.

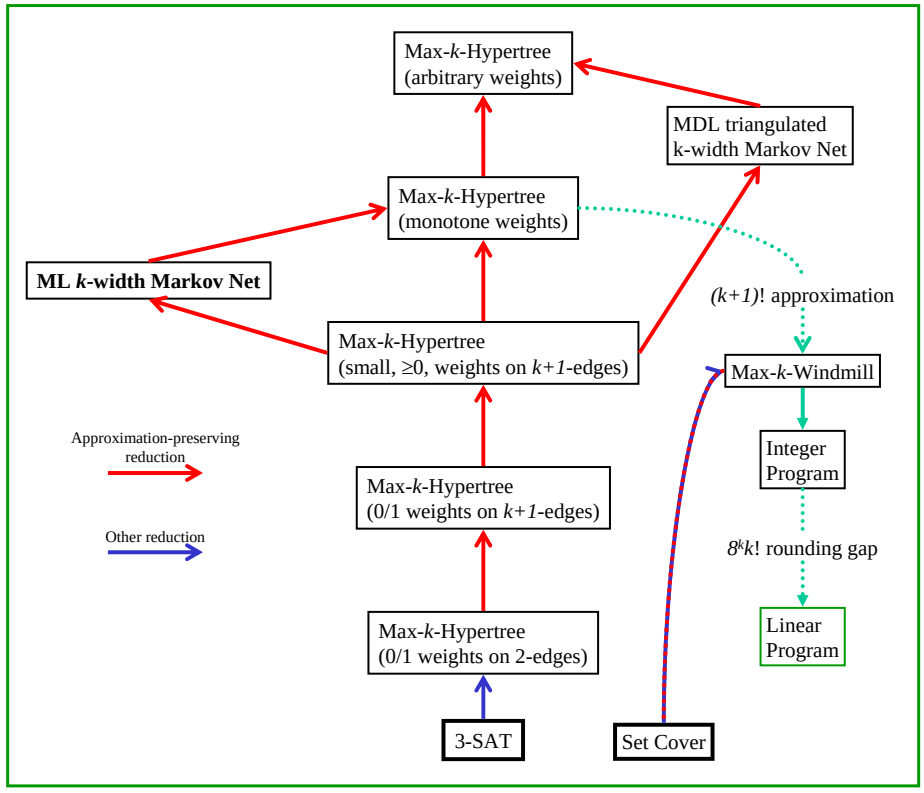
•On $k+1$ candidate cliques, there is a bias of $b' = b \Big/ \binom{n}{k+1}$ resulting in a weight equal to the capacity of a binary bit with such bias:

$$\begin{aligned} w'(h) &= -H(X_h) - \sum_{h' \subset h} w'(X_{h'}) \\ &= \sum_{v \in h} H(X_v) - H(X_h) \\ &= (k+1) + 2^k \tfrac{1+b'}{2^{k+1}} \log \tfrac{1+b'}{2^{k+1}} + 2^k \tfrac{1-b'}{2^{k+1}} \log \tfrac{1-b'}{2^{k+1}} \\ &= 1 - H(\tfrac{1+b'}{2}) = \tfrac{1}{2}((1+b')\log(1+b') + (1-b')\log(1-b')) \end{aligned}$$

•By setting the biases b_h accordingly, we can get weights proportional to our desired weights (they will be very small…).

Uniqueness of factorization

Consider mappings $P_h \to \varphi_h$ from marginal distributions over subsets of variables, to factors over the subset. The mapping giving the factors we suggest is the only such mapping, such that $P_x(x) = \prod_{\text{Cliques } h} \varphi_h(x_h)$ holds for every triangulated graph G and every Markov network P over G.



A Sample Yielding Desired Weights

•We cannot get exact rational weights: we are limited to rational biases, and the bit-capacity function is an irrational function. Instead, we will get weights close enough so that the error (between the ML weights and the desired weights) will be less then the granularity of the weights.

•We approximate the weights using the first term in the Taylor expansion of the bit-capacity function, and bound the error using the second term:

$$w(b) = 1 - H(b) = \sum_{i=2 \text{ even}}^{\infty} \frac{b^i}{i(i+1)}$$

$$\tfrac{1}{2}b^2 < w(b) < \tfrac{1}{2}b^2 + \tfrac{\ln 4 - 1}{2}b^4$$

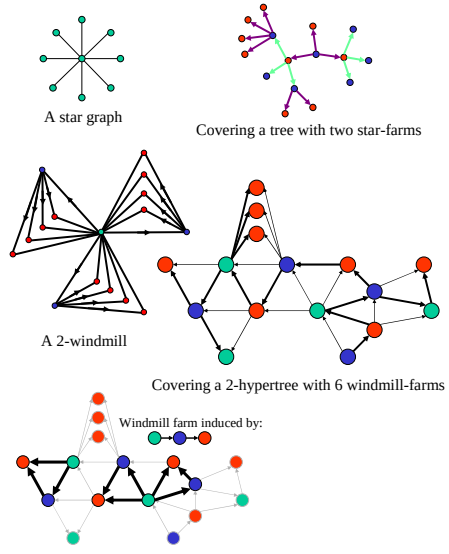
•By scaling down the weights, the granularity scales down linearly while the error scales down quadratically. By scaling down far enough, we can bring the total summed error to be lower than the granularity.

•To build a mixture component P_h with desired bias $b=p/q$, we use q blocks of *(k+1)*-wise uniformly independent sample vectors. But for p of these blocks, we invert the variables of h appropriately so as to set their parity to be odd for all samples in the block.

•Unlike the "pure" distribution, biases of more than $k+1$ variables will be highly biased. However, this does not disrupt the weights. The only important thing is that all k-marginals are uniform, and that for each mixture component, only a single $k+1$-marginal is biased.

•Block of *(k+1)*-wise uniformly independent binary samples can be created of size $2n^{k+1}$ [Alon, Spencer, Erdos 1991], yielding a total sample size of $O(Qn^{2k+2})$, where Q is the common denominator of the rational biases.

Windmill Farms



An Approximation Algorithm

[with David Karger, SODA 2001]

For any k, and an input monotone weight function on cliques, we find a k-hypertree (triangulated graph with clique size at most $k+1$) G, such that the weight $w(G)$ of the graph, is within a (very large) constant factor away from the optimal graph G^*:

$$w(G) \geq \frac{\max_{\text{trig } G^*, \text{width} \leq k} w(G^*)}{8^k k!(k+1)!}$$

Outline of the Algorithm

•A k-hypertree always contains a k-windmill-farm which captures at least 1/(k+1)! of its weight. Instead of searching for the maximal k-hypertree, we'll search for the maximal k-windmill-farm.

•Windmills are easier to work with than hypertrees, since they can be defined in terms of *local* rather then *global* structure.

•We can write down an integer program for the the maximum k-windmill-farm problem.

•We solve a the linear-programming relaxation of the integer program. (Actually, we first have to strengthen it a bit)

•We then iteratively round the linear program to get a integer solution, but at each of k iterations of rounding we might retain as little as $1/8k$ of the weight.

The Maximum Windmill Farm Problem as an Integer Program

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

A rounding scheme

For i=1..k

–For each node v, consider all paths p of length i ending at v, choose path p with probability x_p. And around it to one. The LP assures that the sum of these probabilities is at most one (it might be less, and so we might not choose any path).

–Re-optimize the LP for paths longer than i, using the rounded values for paths of length up to i.

In each iteration the value of the rounded LP is at least 8(k+1-i)! of the value before rounding.