Partitions for Efficient Inference on Graphical Models

Wenlu Hu
Andrew ID: wenluh
wenlu@cmu.edu

Kirthevasan Kandasamy
Andrew ID: kkandasa
kandasamy@cmu.edu

Joseph Tassarotti
Andrew ID: jtassaro
jtassaro@andrew.cmu.edu

Note: Many of the ideas in this paper, as well as significant portions of earlier drafts of the report, were contributed by Elara Willett, who is no longer enrolled in the course.

1 Introduction

Probabilistic Graphical Models such as Bayesian Networks and Markov Random Fields are ubiquitous in Machine Learning and Statistics. They allow us to efficiently represent independence structures among variables in your probability space. Gibbs Sampling, [Liu01] is a Markov Chain based technique for drawing samples from such a graphical model. In Gibbs Sampling, we iteratively sample a subset of variables conditioned on the other variables. We start with an arbitrary state and then run the Gibbs Sampler for a specified burnin time after which the samples are collected. As with most MCMC based methods, samples collected via Gibbs Sampling are correlated. Correlation between samples is an undesirable property [Liu01] and affects our procedure in 2 ways. Firstly, the burn in time will have to be large in order to make the "sample quality". The correlation between samples depends on how slow the Markov Chain (MC) mixes.

A Naive Gibbs Sampler samples one variable at a time. This strategy is computationally simple but usually there is significant correlation between samples. Hence, we might need to run the Gibbs Sampler for several iterations. A commonly used strategy is to group multiple variables together in a Blocked Gibbs Sampler which improves the mixing rate of the MC at the cost of increased computation for sampling from a more complex conditional distribution at each step. Of course, making these blocks arbitrarily complex is not an option since that would make sampling intractable nullifying any gain obtained via a faster mixing time. The first figure of Figure 1 shows a simple graph followed by two ways to block the variables in the graph for a Blocked Gibbs Sampler.

This leads to the interesting question as to whether we could develop an optimal blocking strategy that gives us the best computational tradeoff between mixing time and computational effort. Unfortunately studying this problem is not only too difficult but even not well defined given that computational effort depends on several factors such as the inference algorithm and the platform. However, in the past some work has studied tree partitions [HdF04, HRdF06, Riv05]. While Inference on a general graph is intractable, on trees algorithms such as Belief Propagation converge exactly in a finite number of steps. Hence, in this project we will adopt this strategy. For simplicity we assume that tree partitions provide a good bargain in terms of computation. Hence, we will focus on obtaining the "best" tree-blocks of the graph.

To our knowledge, this problem has not studied extensively in Machine Learning Literature – current work only analyze simpler versions of the problem. We aim to develop a method to work on arbitrary graphs. One of our important novelties will be to take into consideration correlations between variables in order to improve mixing time. While we can make no guarantees about the results we will get, we are excited to be working on what we believe is an interesting and relevant problem.

2 Related Work

Finding partitions of a graph into trees for efficient inference has been studied previously. In [Riv05], they use a very simple heuristic which aims to minimize the number of trees. In [HdF04, HRdF06],

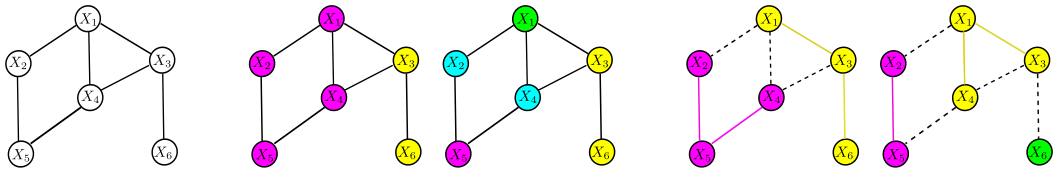


Figure 1: The first figure shows our original graph. The next two figures show to blocking strategies for the variables in the graphs. The final two figures show tree-blocking strategies. Trees are computationally convenient than general blocks since inference on trees is computationally easy.

they show that a Rao-Blackwellised Estimator obtained via conditioning a two-tree partition of a Grid graph outperforms a naive Gibbs sampler.

In order to study this problem methodically, we first need to ask ourselves what characterizes faster mixing in a Markov Chain. [Liu01, LWK94] argue that the mixing rate of a Markov Chain is proportional to the norm of a "Forward operator" F defined on a Hilbert Space of functionals of the variables. Hence a partitioning which has smaller norm in the induced MC can be expected to mix faster. [HRdF06, HdF04] use these results to compare two different partitions of a grid graph. They argue the superiority of one over the other via the norm of the forward operator of the partition and back this conclusion with some experimental results.

3 Theory, Intuition and Insight to our Methods

To set things up for the ensuing discussion, we begin with some results on Markov Chains taken from [Liu01, LWK94]. We are given a Markov Chain $X^{(1)} \to X^{(2)} \to \cdots \to X^{(t)} \to X^{(t+1)} \to \ldots$ with equilibrium distribution π. We define the following set of functionals acting on our probability space $L_0(\pi) = \{h : \Omega \to \mathbb{R} : \quad \mathbb{E}_\pi h(X) = 0, \quad \mathbb{V}_\pi h(X) < \infty\}$ and the inner product $\langle h, g \rangle = \text{Covar}_\pi(h(X), g(X))$. It is straightforward to show that $\left(L_0(\pi), \langle \cdot, \cdot \rangle\right)$ is a Hilbert Space (HS). It can also be shown that the variance of a functional $h \in L_0(\pi)$ is its squared norm: i.e. $\|h\|^2 = \mathbb{V}_\pi(h(X))$. On this Hilbert Space we define the following operator $F : L_0(\pi) \to L_0(\pi), \quad [Fh](\cdot) = \mathbb{E}[h(X^{(1)})|X^{(0)} = \cdot]$. In MC literature F is known as the Forward Operator. For reasons that will become clear later we will call the norm of this operator $\|F\|$ the "Mixing Rate". Using the Law of Total Variance we can show that $\|F\| \leq 1$. Further, if the MC is reversible (which is the case with a Gibbs Sampler) F is self-adjoint and hence the norm of the concatenation of F satisfies $\|F^n\| = \|F\|^n$. [Liu01] gives two results on $\|F\|$. The first of them is characterizes faster mixing.

Theorem 1 ([Liu01] Lemma 12.6.3)**.** *Let $X^{(0)} \sim P^{(0)}$ where $P^{(0)}$ be an initial distribution satisfying some regularity conditions. Let $P^{(n)}$ denote n^{th} step evolution of $P^{(0)}$ obtained by passing it through the Markov Chain n times. Let $\mathbb{E}^{(n)}$ be the expectation under $P^{(n)}$. Then,*

$$|\mathbb{E}^{(n)}h(X) - \mathbb{E}_\pi h(X)| \leq C \|F\|^n \|h\|.$$

In brief, the above theorem says that $P^{(n)}$ will be closer to π at a rate determined by the mixing rate. The second result is on the variance of samples collected after sufficient burn in.

Theorem 2 ([Liu01] Theorem 12.7.2)**.** *Let $x^{(1)}, \ldots, x^{(m)}$ denote samples obtained from the Markov Chain after sufficient mixing and $\bar{h}_m = \frac{1}{m}\sum_{j=1}^{m} h(x^{(j)})$. We then have the following CLT,*

$$\sqrt{m}(\bar{h}_m - \mathbb{E}_\pi[h]) \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma(h)^2) \qquad \textit{where}$$
$$\sigma(h)^2 \leq \mathbb{V}_\pi[h]\left(1 + 2\sum_{j=1}^{\infty}\|F\|^j\right)$$

The above theorem says that the asymptotic variance of the samples collected from a Gibbs Sampler increases monotonically with $\|F\|$. These results convince us that suggest that keeping $\|F\|$ small in a Gibbs Sampler is advantageous to obtaining higher quality samples. But how can this be done ?

Figure 2: The left graph depicts a spanning tree that is not a valid 1-splitting since it violates criteria 2. The right graph depicts a 2-splitting.

[Liu01] also present the following equivalence between $\|F\|$ and the maximal correlation coefficient between samples collected from a MC.

$$\|F\| = \sup_{f,g} \text{Corr}(f(X^{(t+1)}), g(X^{(t)})) \tag{1}$$

Here, $X^{(t)}, X^{(t+1)}$ are successive samples from a Markov Chain and the supremum is over all functionals in $L_0(\pi)$.

While we do not have solid theoretical insight beyond this, we base our methods on the following intuitions gained from above. In a Blocked Gibbs Sampler, correlation between successive samples are induced by having correlation between blocks. Hence, it pays to minimize the correlation between blocks. Equivalently, we could try placing all strongly correlated variables in the same block. (We note that this intuition is not new in the Machine Learning community, but we were not able to find a precise theoretical formulation of this notion.) Now given, a graphical model, we now treat it as a weighted graph with the weights on edges capturing the correlation between the variables at its ends. Our problem now is to find tree partitions of this graph so as to maximize the intra-tree weights.

This leaves us with two challenges, The first is to weight any given graphical model using an appropriate measure of correlation. Generally, this problem is nontrivial. We discuss this issue and present our work around in Section 4.3. The second is to partition a graph into trees given such a weighting. This problem is NP-hard. In Section 4.1, we outline some heuristic algorithms to achieve this objective.

4 Methods

4.1 Tree-Partitioning Algorithms

In order to employ blocked Gibbs sampling on our graphical model, we first partition the underlying graph into blocks. Inference on the blocks must be tractable, so we require the edges between vertices within a block to be a tree. We call the partitioning of a graph into blocks a splitting, which we define precisely as follows.

k-Splitting: Given a graph $G = (V, E)$, parameter k, and a weight function $w : E \to R$, output trees $T_1 = (V_1, E_1), ..., T_k = (V_k, E_k)$ such that:

1. For all $i, j \in [k]$, $V_i \cap V_j = \emptyset$, and $\cup_{i=1}^{k} V_i = V$.
2. For all $i \in [k]$, $u, v, \in V_i$, $\{u, v\} \in E_i$ if and only if $\{u, v\} \in E$.
3. For all $i \in [k]$, $u, v, \in V_i$, there is a unique path from u to v in T_i, i.e. T_i is a tree.

Note that a tree is determined by its vertex set. So we can alternatively think of k-splitting as a coloring of the vertices, where the induced subgraph of each color is a tree. Also notice that the second criteria makes the k-splitting solution distinct from a spanning tree as show in Figure 1.

One possible measure of quality of a splitting is

$$weight(T_1, ..., T_k) = \sum_{i=1}^{k} \sum_{e \in E_i} w(e)$$

Notice that in the case of a unit weight graph, the measure becomes

$$weight(T_1, ..., T_k) = \sum_{i=1}^{k} \sum_{e \in E_i} 1 = \sum_{i=1}^{k} |E_i| = n - k$$

This tells us that in the unit weight case, the weight is maximized by minimizing the number of trees, k. In the design of our algorithms, the intermediate objectives we consider, which are believed correlate to fast convergence are

Unit-weight Objective: Find a k-splitting for a small k.
General-weight Objective: Find a splitting with a small weight.

4.2 Splitting Algorithms

For a baseline, we consider a weighted-edge variant of the splitting algorithm from [Riv05]. For their algorithm, they greedily grow trees favoring vertices of low degree. We present a slightly simplified version below which also tries to incorporate edge weights. The priority queue is defined by **compare(u,v)**:

$$|V \cap N(u)| > |V \cap N(v)| \iff u < v$$

where $N(u)$ is the set of neighbor vertices to u, and V is the set of uncolored vertices. Thus, we favor vertices with fewer uncolored neighbors.

Below is a simplified version of their algorithm that tries to use edge weights.

Greedy Tree Growing Algorithm [Riv05]

1. Initialize $i = 0$ and V to the vertex set.
2. While $V \neq \emptyset$
   - Select $v \in V$
   - Start a new tree T_i and a priority queue Q_i. Add v to Q_i
   - While $Q_i \neq \emptyset$
     - Pop u from Q_i.
     - Initialize $neighborsInT = 0$.
     - For all $v \in T_i$, if $u \in N(v)$, increment $neighborsInT$
     - If $neighborsInT \leq 1$,
       * Add u to T_i and remove v from V
       * Add $N(u)$ to Q_i.
3. Return $\{T_i\}$

Next, we present our contribution, the greedy edge selection algorithm. In our algorithm, we greedily select edges favoring edges of high correlation. To this end, we weight the edges by correlation.

Greedy Edge Selection Algorithm

1. Construct an ordered list of edges, E, with $E[0]$ being the highest weight edge. Edges are vertex pairs (i, j).
2. Initialize an all-zero n-dimensional integer list V of vertex colors.
   ($V[i]$ is the color of vertex i, and $V[i] = 0$ means that vertex i has not yet been colored.)
3. Initialize n empty vertex sets: $T_1, ..., T_n$
   (Logically, T_i is the set of vertices labeled with color i.)
4. Initialize $unusedColor = 1$.

5. For each edge $e = (i, j)$ in E,
   - If $V[i] = V[j] = 0$,
     - Set $V[i] = V[j] = unusedColor$
     - Add i, j to $T_{unusedColor}$
     - Increment $unusedColor$ by 1
   - Else if $V[i] = 0$ and $V[j] \notin getOtherNeighborColors(\{i\}, e)$,
     - Set $V[i] = V[j]$
     - Add i to $T_{V[j]}$
   - Else if $V[j] = 0$ and $V[i] \notin getOtherNeighborColors(\{j\}, e)$,
     - Set $V[j] = V[i]$
     - Add j to $T_{V[i]}$
   - Else if $V[i] \neq 0$ and $V[j] \neq 0$ and $V[i] \notin getOtherNeighborColors(T_j, e)$,
     - For each $k \in T_j$, set $V[k] = V[i]$
     - Set $T_i = T_i \cup T_j$
     - Set $T_j = \emptyset$
   - Otherwise do nothing
6. For each vertex i, if $V[i] = 0$, set $V[i] = unusedColor$, $unusedColor++$
7. Output $\{T_j : T_j \neq \emptyset\}$

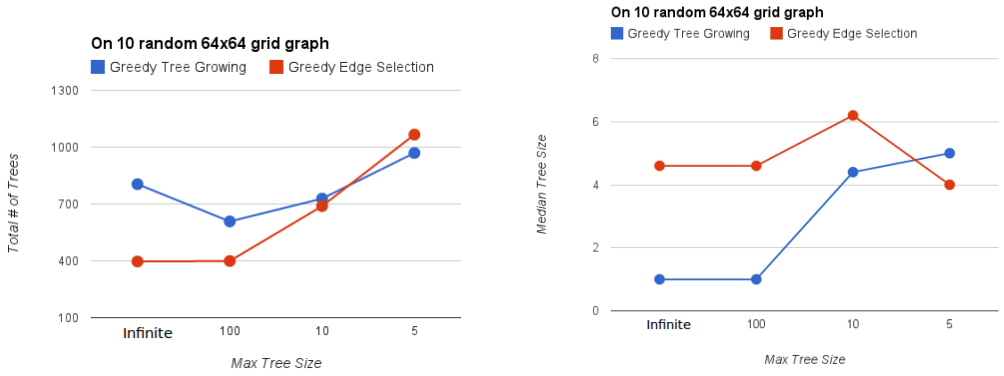
getOtherNeighborColors(S, e)

1. Remove edge e from the graph.
2. Initialize sets $neighbors = \emptyset$, $colors = \emptyset$
3. For each $i \in S$, $neighbors = neighbors \cup N(i)$
4. For each $i \in neighbors$, $colors = colors \cup V[i]$
5. Add e back into the graph.
6. Return $colors$

4.3 Edge Weighting

We now turn to the problem of how we may weight these edges given a graphical model. Generally, this problem is NP-hard. Even for the relatively simple class of log-linear models it is not possible to estimate the correlations between two variables directly. One technique is to sample from the model (say via a Gibbs sampler) and estimate the correlation using the sample – but this obviously defeats the purpose since our very objective is to speed up the Gibbs Sampler. This paints a rather bleak picture of the proposed strategy and at this point, we unfortunately do not have a satisfying solution to it.

But in some cases this information could come via Expert knowledge. For example, in large networks in computational biology etc. expert knowledge is already used to group correlated variables together. In such situations our method is applicable. Next there could be situations in which you repeat the same inference task on a given graphical model under different observations. In such situations we could run a Naive Gibbs sampler on a prototype observation and use it to estimate the correlations. Then on subsequet inference procedures we use a blocked gibbs sampler with tree blocks partitioned using the weights thus learned. Even though observations do play an important role as when conditioned any two variables could become more or less correlated - it is fair to assume that they would be a reasoanable approximation to the actual correlations.



(a) Total number of trees as maximum tree size limit decreases.

(b) Median tree size as maximum tree size limit decreases.

Figure 3: Comparing Greedy Edge Selection algorithm with Greedy Tree Growing algorithm

5 Experiments

5.1 Tree Splitting

We have implemented both tree splitting algorithms described in the Methods section. We have run the algorithms on random-weight grid graphs of various sizes. In these experiments we use random weights as priorities for both algorithms. We have measured the total number of trees generated, and the median tree size, which we believe are correlated to fast convergence.

The results in figure 3(a) and figure 3(b) show that our heuristic performs much better than the weighted edge variant of the splitting algorithm from [Riv05], when no other limits are imposed. The leftmost data points in figure 3(a) show that our heuristic generates half as many trees as the Greedy Tree Growing algorithm. In figure 3(b),we also see that median tree size for our Greedy Edge Selection algorithm is much larger than the other algorithm. The median tree size for Greedy Tree Growing algorithm is only 1, which means at least half of the trees generated have only one vertex each.

Because sampling time can increase as trees grow in size, we are also interested in the properties of the trees generated by the algorithms when we impose a limit on the maximum tree size. However, as we impose a tighter limit on the tree sizes, figure 3(a) and figure 3(b) show that the Greedy Tree Growing algorithm catches up with our heuristic. And as the limit becomes as tight as 5, Greedy Tree Growing algorithm produces slightly fewer trees and a larger median tree size. So our heuristic only outperforms Greedy Tree Growing algorithm when the limit on maximum tree size is no tighter than 10.

5.2 MCMC

To compare how these tree construction heuristics affect inference, we performed a variant of an image reconstruction test found in [HdF04]. In this experiment, a 64×64 pixel test image was converted to grayscale, downsampled to a 4-bit colorspace, and then corrupted by randomly flipping the color of 25% of the pixels. This problem was converted to an undirected graphical model with a grid topology with one node for each pixel and isotropic edge potentials. We then tried to reconstruct the original image using different blocking strategies:

1. Naive Gibbs sampler – no blocking
2. Checker Board blocks – the network was divided up into two blocks in a checker board pattern
3. Hamze-Freitas Two Trees – the network was divided up into two meshed trees

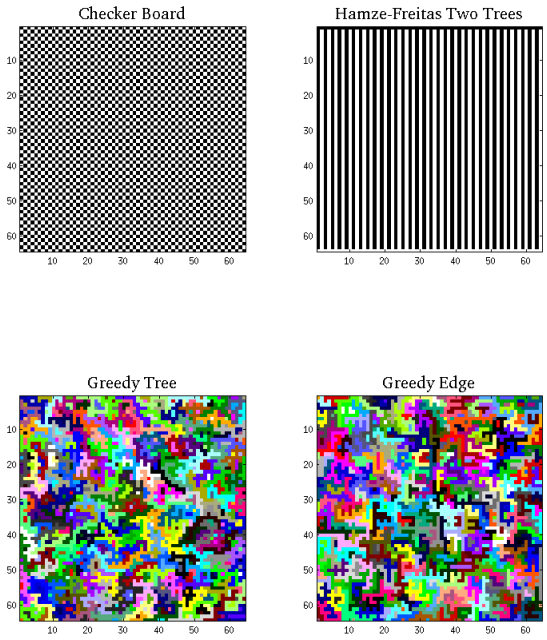


Figure 4: Visualization of different blocks. Each color represents a separate block. The Greedy Edge and Greedy Tree blocks have a maximum tree size of 20. Note that because trees are assigned colores randomly in this visualization, nearby trees may coincidentally be assigned a similar color, making it difficult to differentiate them.

4. Trees generated using Greedy Tree algorithm – we approximated correlations for the model by running 100 iterations of a separate Gibbs sampler. Tree sizes were limited to a maximum of 20.
5. Trees generated using Greedy Edge algorithm – These used the same correlation estimates and tree size maximums.

A visualization of the various blocks is shown in figure 4. The first of these three strategies were compared in [HdF04]. Our experiments were conducted using Schmidt's MATLAB library for undirected graphical models [Sch07], which we modified to add in the Rao-Blackwellized tree sampler described in [HdF04]. Each of the five MCMC chains for each trial were started from the same initial point.

The results of this experiment are found in figure 5. The relative performance gap between the Naive, Checker Board, and Two Trees are in line with the results in [HdF04]. The trees generated by the two heuristics perform in between the Checker Board and the Two Trees, with the Greedy Edge performing slightly better. Note that these experiments plot the error rate versus the number of iterations of the Gibbs sampler, rather than total computation time. In practice, the Two Trees strategy and the automatically generated trees were considerably slower than the Checker Board and the Naive Gibbs sampler, but this may be an artifact of the implementation we used.

6 Conclusion

We proposed a method to develop tree partitions for efficient blocked Gibbs Samplers. The primary idea was to weight the edges of the graphical model according to the correlation of the two variables and then convert it into a tree partitioning problem. The objective in tree partitioning is to preserve the high weight (highly correlated) edges within the trees and remove the low weight edges. We outlined two heuristics to perform this which performed reasonably well on our experiments.

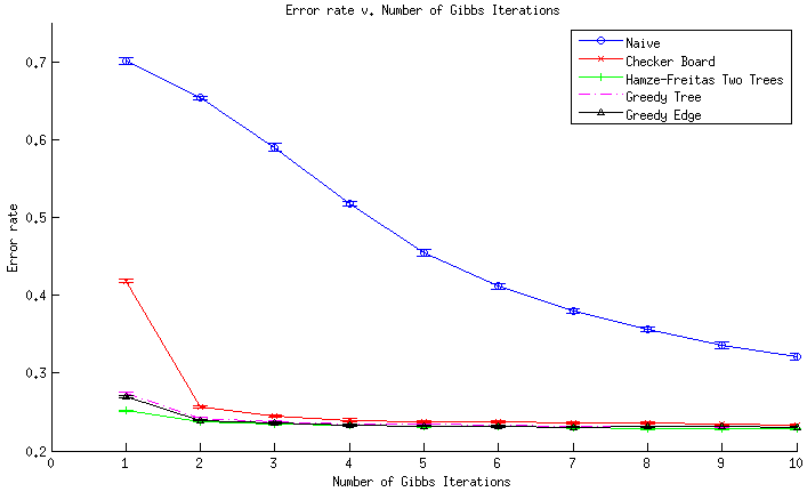


Figure 5: Comparison of the performance of the Naive Gibbs, Checker Board, Hamze-Freitas Two-Trees, and automatically generated trees using the Rao-Blackwellized sampler on a vairant of the image reconstruction task from [HdF04].

Yet there are several points of concern in our work. The first is that there is no straightforward way to estimate the correlation between two variables in a graphical model without performing an actual inference task itself. At best, we only know of some cheap ways to approximate this. However, trees partitioned this way could still work well in practice. Second, while we did achieve faster mixing in terms of iterations, when you factor in computation time the Naive Gibbs Sampler beats tree sampling since it is computationally much cheaper. With faster tree sampling implementations, we hope that we would be able to break this barrier though. Third, in many graphical models especially sparse high dimensional ones - the tree structure encodes a significant amount of the dependence structure. That is, two variables connected by an edge in a graphical model are quite likely to be correlated themselves. In such situations, it is not clear that our method would significantly outperform (say) a random tree partitioning strategy.

Acknowledgements

We thank Elara Willet for helping us with the graph theory element of the project. She has worked closely with us in the first half semester, in particular with the design of the tree generation heuristics.

References

[HdF04] Firas Hamze and Nando de Freitas. From Fields to Trees. In *UAI*, pages 243–250, 2004.

[HRdF06] Firas Hamze, Jean-Noel Rivasseau, and Nando de Freitas. Information theory tools to rank MCMC algorithms on probabilistic graphical models. In *Proceedings of the UCSD Workshop on Information Theory and its Applications Invited Paper, San Diego, CA*, 2006.

[Liu01] Jun S. Liu. *Monte Carlo strategies in Scientific computing*. Springer, 2001.

[LWK94] J. S. Liu, W. H. Wong, and A. Kong. Covariance structure of the Gibbs sampler with applications to the comparisons of estimators and augmentation schemes. *Biometrika*, 81(1):27–40, March 1994.

[Riv05] Jean-Noël Rivasseau. *From the jungle to the garden: Growing trees for Markov Chain Monte Carlo inference in undirected graphical models*. PhD thesis, Citeseer, 2005.

[Sch07] Mark Schmidt. UGM: Matlab Code for Undirected Graphical Models. http://www.di.ens.fr/ mschmidt/Software/UGM.html, 2007.