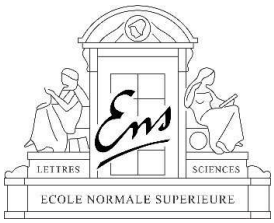
Invert sparse Matrices with Gaussian Belief Propagation

Pierre Gaillard
Département d'Informatique
École Normale Supérieure
Email : pierre.gaillard@ens.fr

Devavrat Shah
Laboratory for Information and Decision Systems
Massachussett Institute of Technology
Email : devavrat@mit.edu



September 6, 2010

Abstract

The Gaussian Belief Propagation (GaBP) algorithm is a well know algorithm that gives beliefs of marginal distributions in Gaussian Markov Random Fields (GaMRF). If the graph is singly connected the beliefs are exact. If it is not, the algorithm converges only under some conditions like Walk-summability or diagonally dominant property. The beliefs of the means of the marginal distributions are then proved to be still exact but the variances are not. We are interested in how we can use the estimations made by the algorithm to get an estimation of the covariance matrix in the field. It will in fact be an estimation of the invert of the matrix corresponding to the graph. The aim here is how to use GaBP algorithm so as to invert matrices and how efficient it would be ? We will see that we do not actually have to be limited to GaMRF. We can use this idea to invert any diagonally dominant matrix approximately.

1 Gaussian Belief Propagation

The GaBP algorithm is usually used on GaMRF. It gives beliefs of the marginal distributions in the field with an efficient way based on passing local messages.

1.1 Introduction to Gaussian Belief Propagation in Markov Random Fields

In this section, we present the GaBP algorithm and how it is used to solve inference problems in GaMRF. You may find more details in [1, 2].

Let $(x_i \in V)$, a set n random variables verifying the join probability

$$p(x) \sim exp(-\frac{1}{2}x^T Ax)$$

where A is a symmetric definite positive matrix. Then we can define a graphical model $G = (V, E)$ describing this correlation between the nodes. V is the set of nodes and corresponds to the random variables x_i and E is the set of edges and describes the correlations between this variables. It includes all not-zero entries of A for which $i > j$. We can rewrite the density function to emphasize the pairwize nature of distribution

$$p(x) \sim \prod_{\{i,j\}\in E} exp(-\frac{1}{2}x_i A_{ij} x_j) \prod_{i\in V} exp(-\frac{1}{2}x_i^2 A_{ii}).$$

In the GaBP algorithm, we introduce new scalars such as P_{ij}^t which can easily be understood as a message from the node x_i to the node x_j about what state j should be in (see figure 1) after t iterations. At the end of the algorithm, through the messages coming from their neighbors, each node has easily access to an estimation of his variance. We can remark that all the marginal distributions in the network are still Gaussian and that there mean are zero. Hence the variances of the nodes or the covariance matrices between nodes are enough to get marginal distributions. The final estimation of the variance of x_i will be

$$B_{ii}^d = \frac{1}{A_{ii} + \sum_{j\in N(i)} P_{ji}^t} \tag{1}$$

We also can get an estimation of the covariance matrix between two neighboring nodes x_i and x_j,

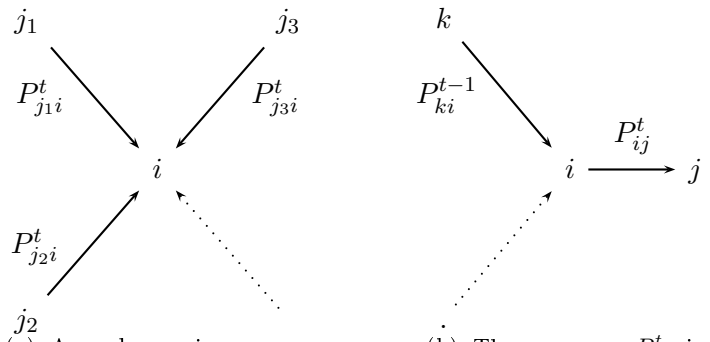
$$B_{ij}^d = \begin{pmatrix} A_{ii} + \sum_{k\in N(i)\setminus j} P_{ki}^t & A_{ij} \\ A_{ji} & A_{jj} + \sum_{k\in N(j)\setminus i} P_{kj}^t \end{pmatrix}^{-1}_{12} \tag{2}$$

where $N(i)$ denotes the nodes neighboring i. The idea is that the message P_{ij} says to the node j the uncertainty caused by his correlation with the

subgraph coming from the node i. We can define the message update rules that follow the same idea. We initialize $P^0 = \text{zeros}(n)$. For all $t \geq 0$ and $i \neq j \in V$,

$$P_{ij}^t = -\frac{A_{ij}A_{ji}}{A_{ii} + \sum_{k \in N(i)-j} P_{ki}^{t-1}} \tag{3}$$

The estimations of the variances are exact if G is a tree, ie if it is singly connected. However if the graph contains loops, there is redundant information and there will be mistakes. Note that the updates rules of message passing can be generalized to MRF that are not Gaussian.



(a) A node receives messages from all his neighbors.

(b) The message P_{ij}^t is calculated according to the update equation 3.

Figure 1: The message passing scheme of Gaussian Belief Propagation

1.2 Invertion of more general trees with GaBP

In this section we show how to derive the GaBP algorithm for inversing directed trees, ie the matrix corresponding to the tree. A directed tree T is a graph such that his adjacency matrix is a subgraph of an undirected tree. In the previous section the matrix had to be symmetric definite positive. However simulations show that the algorithm still works with not symmetric matrices following the same message passing scheme.

We give here an algebraical view of the algorithm in trees in order to explain why it allows to invert them. We will then not deal with distributions, marginal distributions, or MRF.

Actually GaBP works with every directed or not graph $G = (V, E)$ such that there is at most one direct path between two nodes. The graph must then avoid loops. Trees for instance hold that property but the graph is not necessary a tree.

Let A be the matrix related to G. As it exists a bijection between matrices and graphs, I use these two terms without distinguish them. We are going to

show that using gaussian belief propagation message update rules 3 allows us to invert A.

What is the intuition behind this rules and why do they allow to invert A when G is a tree ? It is based on the following idea. Each step of the algorithm simplify the inversion of the matrix by erasing the leaves. Suppose that A corresponds to a tree and that the first node is a leaf and the second his only neighbor. A looks like,

$$A = \begin{pmatrix} A_{11} & A_{12} & 0 & \cdots & 0 \\ A_{21} & & & & \\ 0 & & & & \\ \vdots & & A_{\setminus 1} & & \\ 0 & & & & \end{pmatrix}.$$

Simple algebraical properties give that the invert $B = A^{-1}$ of A verifies

$$B = \begin{pmatrix} \star & \star & \cdots & \star \\ \star & & & \\ \vdots & & B_{\setminus 1} & \\ \star & & & \end{pmatrix}$$

with $B_{\setminus 1} = \text{inv}(A_{\setminus 1})$ where we modified the diagonal value $A_{\setminus 1}(1,1) \leftarrow A_{\setminus 1}(1,1) - \frac{A_{12}A_{21}}{A_{11}}$. The algorithm uses this idea of simplification and does it efficiently and simultaneously in the graph.

After d iterations, all the subtrees of depth less then d can be simplified. This can be show by a simple induction. The algorithm by adding the values of the messages coming from the subtrees that we want to eliminate to the diagonals values can give us the smaller matrices that we have to invert, to get a sub-matrices of B. After $\text{diam}(A) - 1$ iterations, we just have 2x2 matrices to invert and we get the diagonal values of B and the off diagonal values B_{ij} when i and j are neighbors.

To fill the other values of B we can do a DFS and use the simple rule for trees, $B_{ik} = \frac{B_{ij}B_{jk}}{B_{jj}}$ where j belongs to the only path between i and k. The issue with the graphs that have loops is that there can be several paths between i and j, and then this rule does not hold any more.

2 Invert sparse random matrices with GaBP

We saw that GaBP can be use to invert matrices related to trees. However GaBP rules do not always converge if the graph has loops. Even if it converges, it does not converge to the invert of the matrix. [2] proved that diagonaly dominant was suffisant condition for convergence. [4] improved

the condition to Walk-Summability. Here, we are going to study how GaBP behaves with diagonally dominant random matrices. The matrix must also be enough sparse to get a good complexity.

We can now define our model and our goal. We have a random graph $G = (V, E)$ with n nodes that is c-regular or Erdös-Renyi $G(n, \frac{c}{n})$ and a matrix A such that

- $A_{ij} \neq 0 \Leftrightarrow (i,j) \in E$
- $\forall i \in V \quad A_{ii} \neq 0$
- $\exists \gamma > 1 \quad \forall i \in V \quad |A_{ii}| \geq \gamma \quad \sum_{j \in V - \{i\}} |A_{ij}|$.

We want to invert approximatively A with the GaBP algorithm, to study how good the approximation is and what is the complexity. We normalize A by left multiplying for instance with D^{-1}, where $D = \text{diag}(A)$.

$$A \leftarrow D^{-1}A.$$

The diagonally dominant property still holds. We note B the invert of A.

All the proofs will be based on the following lemma. The intuition is that the entries of an invert can be see as sum of weighted paths in a graph. The algorithm will do that sum but may forget some paths and his estimation will be false. Hence, if we want to bound the error of the algorithm or the entries of the invert, we can bound the sum of some set of weighted paths. That is exactly what this claim will do. Let be more precise. If we define the matrix $R = I - A$, then $B = A^{-1} = (I - R)^{-1} = \sum_{k \geq 0} R^k$. Then $B_{ij} = \sum_{k \geq 0} R^k_{ij}$ and corresponds to all the weights of the paths between i and j.

$$B_{ij} = \sum_{\omega \in W_{ij}} \rho(\omega) \tag{4}$$

where W_{ij} is the set of the paths between i and j and $\rho(\omega) = \prod R_{\omega_i \omega_{i+1}}$.

Claim 1. *For all $i \in V$, for all $U \subset W_i$,*

$$\sum_{\omega \in U} |\rho(\omega)| \leq \frac{\gamma^{-d(U)}}{1 - \gamma^{-1}}$$

where W_i is the set of paths leaving from i and $d(U) \hat{=} \min_{\omega \in U} |\omega|$.

Proof. We show this inequality by induction on $d(U)$. For U a set of paths, we note $\rho(U) \hat{=} \sum_{\omega \in U} |\rho(\omega)|$ and we define the property for $k \geq 0$,

$$\mathcal{P}_k \hat{=} \text{"}\forall i \in V \quad \forall U \in W_i \quad st. \quad d(U) \geq k \quad \rho(U) \leq \frac{\gamma^{-k}}{1 - \gamma^{-1}}\text{."}$$

- $\mathcal{P}_0$ **:** First we remark that for all $i \in V$ and $U \in W_i$,

$$\begin{aligned} \rho(U) &\leq \sum_{\omega \in W_i} |\rho(\omega)| \\ &= \sum_{j \in V} \bar{B}_{ij} \end{aligned}$$

where $\bar{B} = \bar{A}^{-1}$ and $\bar{A}$ is the matrix such that $|\bar{A}| = A$ and $\bar{R} = I - \bar{A} \geq 0$. All the weighs of the paths will then be positive. Hence, it exists $M_i \geq 0$ such that for all set of paths $U \in W_i$, $\rho(U) \leq M_i$. If we take $M = \max_{i \in V} M_i$, then

$$\forall i \in V,\ \forall U \in W_i,\ \rho(U) < M.$$

Let $i \in V$ and $U \in W_i$,

$$\begin{aligned} \rho(U) &= \sum_{\omega \in U} |\rho(\omega)| \\ &\leq 1 + \sum_{j \in N(i)} |R_{ij}| \sum_{\omega \in U_j \subset W_j} |\rho(\omega)| \\ &\leq 1 + \sum_{j \in N(i)} |A_{ij}| M_j \\ &\leq 1 + \gamma^{-1} M \end{aligned}$$

1 corresponds to the weight of the empty path. Hence, $M < \frac{1}{1-\gamma^{-1}}$ and $\mathcal{P}_0$ is proved.

- $\mathcal{P}_k$ **:** we assume P_{k-1} is verified. Let $i \in V$ and $U \in W_i$ with $d(U) \geq k$. Similarly as before, we write

$$\rho(U) = \sum_{j \in N(i)} |A_{ij}| \sum_{\omega \in U_j} |\rho(\omega)|$$

where for all $j \in N(i)$, $U_j \subset W_j$ and $d(U_j) \geq d(U) - 1 \geq k - 1$. Hence, according to P_{k-1},

$$\begin{aligned} \rho(U) &\leq \sum_{j \in N(i)} |A_{ij}| \frac{\gamma^{-(k-1)}}{1 - \gamma^{-1}} \\ &\leq \frac{\gamma^{-k}}{1 - \gamma^{-1}} \end{aligned}$$

□

Now we aim to express the error made by the algorithm as a sum of enough long path. The bound will then follow directly.

2.1 The inverse approximate

To get its approximate, the algorithm does the sum of weighed paths but forget some. To quantify its error we should then understand what paths it forgets and what paths are considered by the algorithm.

2.1.1 The considered paths

The algorithm works in two steps. First it sends messages through the edges following the GaBP rules. After d iterations, it can calculate B^d a matrix that gives estimations of the diagonal values of B and of B_{ij} when i and j are neighbors using the equations 2 and 1.

The algorithm acts as if the graph was a tree rooted in the edge (i, j) and of depth smaller than d. He adds the messages P^d_{ki} to A_{ii} so as to simplify the subtrees coming from $k \neq j$ in i (see equations 1 and 2).

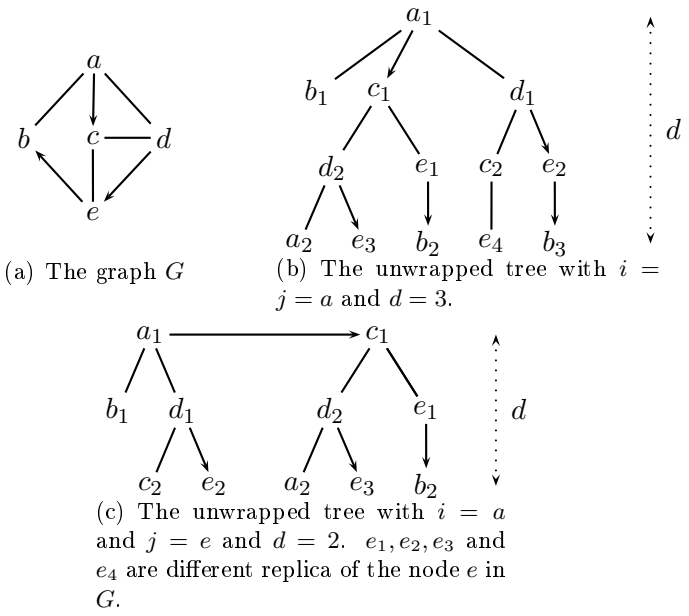
Each estimation B^d_{ij} corresponds to an entry $\tilde{B}(i,j)_{ij}$ of an exact invert $\tilde{B}(i,j)$ of a tree $\tilde{A}(i,j)$. That will allow us to express the estimations sum of weighted paths.

These trees are the computation trees that the algorithm see when it go through the graph. We name them unwrapped trees. Let define the construction of the unwrapped trees $\tilde{A}(i,j)$ when $i = j$ of $j \in N(i)$. We start with the root node i if $j = i$ of with the edge (i, j) if $j \in N(i)$. Then we add all the neighbors of i except j as children of i. We do the same with j. Then we iterate the following procedure $d-1$ times :

- Find all leaves of the tree
- For each leaf l, find $N(l)$ all k_l nodes in the loopy graph that neighbor the node corresponding to this leaf.
- Add $k_l - 1$ nodes as children to each leaf, corresponding to all neighbors but the parent node.

To each node in the unwrapped tree corresponds one node in the original loopy graph G. To avoid confusion, we note with $\tilde{}$ the unwrapped quantities and we note the previous relation $\tilde{l} \sim l$ if $l \in G$ corresponds to $\tilde{l} \in \tilde{G}$. The matrix $\tilde{A}(i,j)$ is then filled verifying $\tilde{A}(i,j)_{\tilde{k_1}\tilde{k_2}} = A_{k_1 k_2}$ where $\tilde{k_1} \sim k_1$ and $\tilde{k_2} \sim k_2$.

We can remark that the messages received in i and j (the root nodes) after d iteration from the other neighbors are exactly the same in the unwrapped tree and in the loopy graph G. We deduce that the estimation B^d_{ij} and $\tilde{B}^d_{ij}$ will be the same and since $\tilde{A}$ is a tree, the estimate $\tilde{B}^d_{ij}$ corresponds to the exact invert $\tilde{B}_{ij}$. Hence, we can write B^d_{ij} as a sum of weighted paths



(a) The graph G

(b) The unwrapped tree with $i = j = a$ and $d = 3$.

(c) The unwrapped tree with $i = a$ and $j = e$ and $d = 2$. e_1, e_2, e_3 and e_4 are different replica of the node e in G.

Figure 2: Construction of two unwrapped trees.

$$\begin{aligned} B_{ij}^d &= \tilde{B}_{ij} \\ &= \tilde{A}_{ij}^{-1} \\ &= (I - \tilde{R})_{ij}^{-1} \\ &= \sum_{\tilde{\omega} \in \tilde{W}_{ij}^d} \tilde{\rho}(\tilde{\omega}) \end{aligned}$$

where $\tilde{W}_{ij}^d$ are the paths in the unwrapped tree from the root node i to the other root node j. Using the correspondence between the unwrapped tree and the original graph, to each $\tilde{\omega} \in \tilde{W}_{ij}^d$ corresponds a unique path $\omega \in W_{ij}$. Furthermore, we have $\rho(\omega) = \tilde{\rho}(\tilde{\omega})$. Hence it exists a set $U_{ij} \subset W_{ij}$ such that

$$B_{ij}^d = \sum_{\omega \in U_{ij}^d} \rho(\omega) \tag{5}$$

where $U_{ij}^d \subset W_{ij}$ denotes the paths from i to j that correspond to paths in the unwrapped tree $\tilde{A}(i, j)$.

2.1.2 The local corrections

As a second step, we do local corrections to consider the small loops and to have approximation of the entries B_{ij} when i and j are not far but not necessary neighbors.

It consists in n Deep First Search of depth d from each node. We use the idea that in trees, if k belongs to the only direct path from i to j then $B_{ij} = \frac{B_{ik}B_{kj}}{B_{kk}}$. However, if the graph has loops, then we may have many possible paths from i to j. We name C^d this second approximation of B and we define it as follow. For $i, j \in V$,

$$C^d_{ij} \;\hat{=}\; \sum_{\substack{i_0 = i, ..., i_k = j \\ 0 \leq k \leq d, \;\; \forall l \; i_{l-1} \neq i_{l+1}}} \frac{\prod_{l=0}^{k-1} B^d_{i_l i_{l+1}}}{\prod_{l=1}^{k-1} B^d_{i_l i_l}}$$

You can find an example of this summation figure 3.

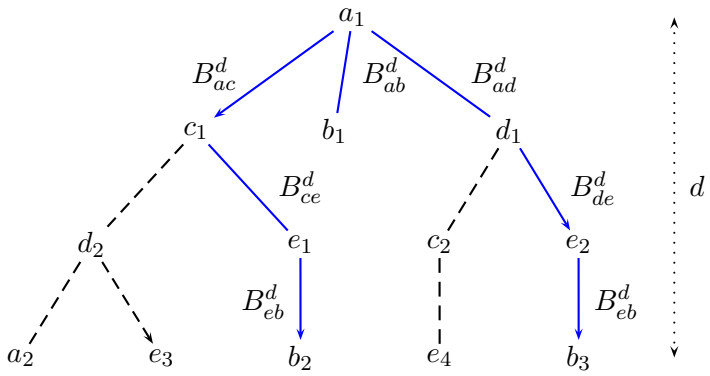


Figure 3: $C^d_{ab} = B^d_{ab} + \frac{B^d_{ac}B^d_{ce}B^d_{eb}}{B^d_{cc}B^d_{ee}} + \frac{B^d_{ad}B^d_{de}B^d_{eb}}{B^d_{dd}B^d_{ee}}$

We can now use the equation 5, to get the following equality.

$$\begin{aligned} C^d_{ij} &= \sum_{\substack{i_0 = i, ..., i_k = j \\ 0 \leq k \leq d \\ \forall l \; i_{l-1} \neq i_{l+1}}} \; \sum_{\substack{\omega \in W_{ij} \\ \omega = \omega_0 ... \omega_{k-1} \\ \forall l \;\; \omega_l \in U^d_{i_l i_{l+1}}}} \rho(\omega) \\ &= \sum_{\omega \in V^d_{ij}} \rho(\omega) \end{aligned}$$

with

$$V^d_{ij} = \left\{ \omega \in W_{ij} \;\middle|\; \begin{array}{cccc} \exists \; i_0 i_1 .. i_k \; \in W^d_{ij} & \text{with} & \forall l & i_{l-1} \neq i_{l+1} \\ \omega = \omega_0 ... \omega_{k-1} & \text{with} & \forall l & \omega_l \in U^d_{i_l i_{l+1}} \end{array} \right\}$$

where W_{ij}^d denotes the subset of W_{ij} including only the paths of length lower then d.

We now want to bound the difference between B_{ij} and C_{ij}^d. If we say that $U_{ij} = \emptyset$ when $d(i,j) > 1$ and $V_{ij}^d = \emptyset$ when $d(i,j) > d$, then we can resume the situation. For all $i, j \in V$ we have

$$\begin{aligned} B_{ij} &= \sum_{\omega \in W_{ij}} \rho(\omega) \\ B_{ij}^d &= \sum_{\omega \in U_{ij}^d} \rho(\omega) \\ C_{ij}^d &= \sum_{\omega \in V_{ij}^d} \rho(\omega) \end{aligned}$$

with $U_{ij}^d \subset V_{ij}^d \subset W_{ij}$. W_{ij} is actually the set of all paths in G between i and j. U_{ij}^d corresponds to the paths in G that correspond to paths in the related unwrapped tree. V_{ij}^d is the set of paths in G that are concatenations of paths in unwrapped tree of depth d rooted in the edges in a path between i and j.

2.2 A bound of the error

We finally expressed the estimation C_{ij}^d of B_{ij} as a sum of some weighted paths. Then using the claim 1, the bound of the error will follow.

$$B_{ij} - C_{ij}^d = \sum_{\omega \in W_{ij} \setminus V_{ij}^d} \rho(\omega)$$

Or,

$$\left| B_{ij} - C_{ij}^d \right| \leq \sum_{\omega \in W_{ij} \setminus U_{ij}^d} |\rho(\omega)| \tag{6}$$

Claim 2.

$$\left\| B_i - C_i^d \right\|_1 \leq \frac{\gamma^{-d}}{\gamma - 1}$$

where $\|X\|_1 = \sum_i |X_i|$.

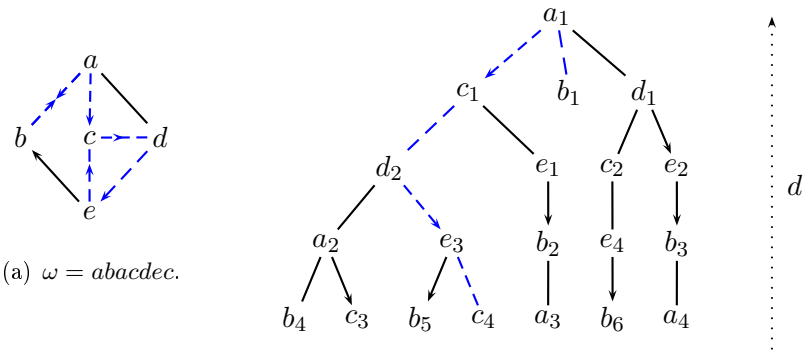
Proof. Let define $\bar{W}_i^d$ the paths leaving from i that are forgotten after d iteration of the algorithm. For all $j, k \in V$, $V_{ij}^d \subset W_{ij}$ and as $(W_{ij})_{j \in V}$ is a partition of W_i,

$$\bar{W}_i^d = W_i \setminus \bigcup_{j \in V} V_{ij}$$

We then remark that all paths in W_{ij} of length smaller then d are in V_{ij}^d, ie $W_{ij}^d \subset V_{ij}^d$. Hence, $d(\bar{W}_i) > d$ and we conclude using the claim 1,

$$\begin{aligned}\left\|B_i - C_i^d\right\|_1 &\leq \sum_{\omega \in \bar{W}_i} |\rho(\omega)| \\ &\leq \frac{\gamma^{-d}}{\gamma - 1}.\end{aligned}$$

Let prove the inclusion $W_{ij}^d \subset V_{ij}^d$. Let $\omega \in W_{ij}^d$. As $|\omega| \leq d$, ω corresponds to a path in the unwrapped tree rooted in i from the root to $\tilde{j}$ a replica of j. Let $\tilde{i_0}\tilde{i_1}...\tilde{i_k}$ the smallest path from the root to $\tilde{j}$ in this tree. There is a corresponding path in the graph $i_0 i_1 ... i_k$. By construction of the unwrapped tree and as is is a smallest path, we have for all $1 \leq l \leq k-1$ $i_{l-1} \neq i_{l+1}$. Finally, we can find a decomposition of ω such that the second condition of V_{ij}^d holds with $i_0, ..., i_k$. The figure 4 gives an example and convinces us easily. Hence, $\omega \in V_{ij}^d$.



(a) $\omega = abacdec$.

(b) The corresponding path in the unwrapped tree is $\tilde{\omega} = a_1 b_1 a_1 c_1 d_2 e_3 c_4$. The smallest path is then $a_1 c_1 d_2 e_3 c_4$.

Figure 4: In the graph G in figure 4(a), we choose a path ω between two nodes a and c. The figure 4(b) shows the corresponding path in the unwrapped tree. We then get the decomposition of ω, aba, c, d, e, c that proves $\omega \in V_{ac}^d$ with $d \geq 7$.

□

2.2.1 The case of not normalized matrices

We did the analysis for normalized diagonally dominant matrices. But the algorithm also works when the matrix A is not normalized. Let $\hat{A} = D^{-1}A$ denotes the normalization of A where $D = \text{diag}(A)$, C^d the direct estimation of A without normalizing and $\hat{C}^d$ the estimation of $\hat{A}$. Then we can show

that $C^d = \hat{C}^d D^{-1}$. We can resume,

$$\begin{aligned} \hat{A} &= D^{-1}A \\ \hat{B} &= BD \\ \hat{C}^d &= C^d D \end{aligned}$$

Hence, we can calculate and bound the error for not normalized matrices.

$$\begin{aligned} \left\| B_i - C_i^d \right\|_1 &= \left\| \left[\hat{B}D^{-1}\right]_i - \left[\hat{C}^d D^{-1}\right]_i \right\|_1 \\ &= \sum_{j=1}^{n} \left| \frac{\hat{B}_{ij} - \hat{C}_{ij}^d}{A_{jj}} \right| \\ &\leq \frac{\gamma^{-d}}{(\gamma-1)\min|\mathrm{diag}(A)|} \end{aligned}$$

If the matrix is symmetric, we can improve it,

$$\left\| B_i - C_i^d \right\|_1 \leq \frac{\gamma^{-d}}{(\gamma-1)\,|A_{ii}|}.$$

2.3 Complexity Analysis

We now study the complexity of the algorithm in time. We count the number of simple operations. The algorithm works in two steps. The first one consists in sending messages through each directed edges during d iterations. We get then $\mathcal{C}_1 = O(ed)$ where e is the number of edges.

Then, as a second step, it does n Deep First Search of depth d. This is the limiting step that take time. The complexity is here $\mathcal{C}_2 = O(\Phi(G))$ where

$$\Phi(G) \hat{=} \sum_{i \in V} \left| G_i^d \right|$$

with G_i^d the unwrapped tree rooted in i of depth d. We can act on this complexity so that $\Phi(G) = O(n^2)$. We will then get the best precision that we can reach in the minimal complexity $O(n^2)$ as we use at least $O(n^2)$ to read the matrix.

Let give some concrete values for random regular graphs. Let first G be a Regular graphs of degree c. we can find two constant $K_1, K_2 > 0$ such that

$$\begin{aligned} \mathcal{C}_1 &= O(ncd) \\ K_1 n(c-1)^d &\leq \Phi(G) \leq K_2 n c^d \end{aligned}$$

Hence, if we take $d = \frac{\ln n}{\ln c}$, the complexity will be $O(n^2)$ and the precision for a whole row $\frac{\gamma^{-d}}{\gamma-1} = O\left(n^{-\frac{\ln n}{\ln c}}\right)$. For Erdös-Renyi graphs, the complexity is very similar. We will do some simulations in the next section to compare.

It can be interesting to compare our algorithm to a naive algorithm that computes with brutal force $\sum_{\omega \in W_{ij}^d} \rho(\omega) = \sum_{k=0}^{d} R^k$. That one will have obviously the same precision bound $\frac{\gamma^{-d}}{\gamma - 1}$ according to claim 1 and a complexity very similar.

In fact, it will have exactly the same complexity for a matrix such that $A_{ij} \neq 0 \Rightarrow A_{ji} = 0$. That is actually not surprising since if the matrix has that property then all the messages will be null during the first step and the second step will exactly be the naive algorithm. Our algorithm is better when the graph has many edges such that $A_{ij} \neq 0$ and $A_{ji} \neq 0$ like undirected graphs. It does the summation without doing any half-turn.

For example, with an undirected random c-regular graph. The naive complexity will by $O(c^d)$ when ours will be $O((c-1)^d)$. Furthermore, if the bound of the precision if the same, our algorithm can consider more paths in the summation and his precision will be better. We will analyse it more precisely in the following section with simulations.

Finally we can remark that we can do convergence acceleration with Aitken method in order to improve some estimates C_{ij}^d. It takes $O(d^2)$ operations and hence we cannot practise it for every entry in the matrix. It would be too expensive. However, it can be nice if we want with accuracy only some estimates and approximatively the others.

3 Simulation Results

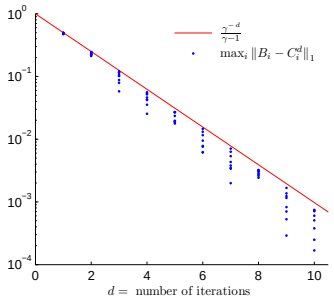
In this section we compare our results with simulations. We did them in **Matlab** and on ER or regular random graphs. We filled the entries with different distributions : Gaussian, uniform,...

3.1 Accuracy of the bound

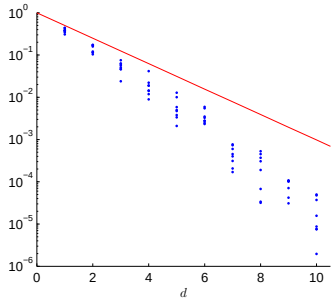
We first test our bound to see if it is accurate or if the algorithm is more precise. We create normalized random matrices of fixed size n by choosing an edge (i, j) with probability p small. We filled the entries with Gaussian variables and we choose the diagonal values in order to verify the diagonally dominant property with constant γ for every row.

We can observe in figure 5(a) that for not symmetric matrices and if $R = I - A \geq 0$, the bound is very accurate. That is not very surprising. Let see what reasons cause the difference between the bound and the real error. If we read again the reasoning to get the bound, we can find three overvaluation inequalities that can be strict.

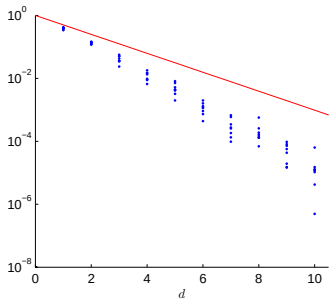
- The first overestimation is in the diagonally dominant property of the matrix A. We can have for some $i \in V$, $|A_{ii}| > \gamma \sum_{j \neq i} |A_{ij}|$. The bound in claim 1 will not be reached.



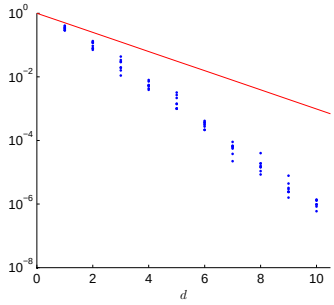
(a) A not symmetric, $R > 0$



(b) A not symmetric, R not positive



(c) A symmetric, $R > 0$



(d) A symmetric, R not positive

Figure 5: Accuracy of the bound for different types of random matrices A with $R = I - A$.

- The second one is in inequality 6, $\left|\sum_{\omega \in W_i} \rho(\omega)\right| < \sum_{\omega \in W_i} |\rho(\omega)|$. If the weighs of the paths $\rho(\omega)$ for ω a path in G are not all positive, ie when $A_{ij} > 0$ for some $i \neq j \in V$ then the inequality will be strict.

- Finally, the last difference occurs when $V_{ij}^d \neq W_{ij}^d$ in the proof of claim 2. We already said that if it does not exist i and j two different nodes in V such that $A_{ij} \neq 0$ and $A_{ji} \neq 0$, then we have $V_{ij} = W_{ij}$. The reverse also holds. Hence, if the matrix is symmetric, this difference will be the biggest in average. The bound cannot be improve but we can better bound the expectation of the error.

We can observe in figure 5 how this overestimations influence the accuracy of the bound.

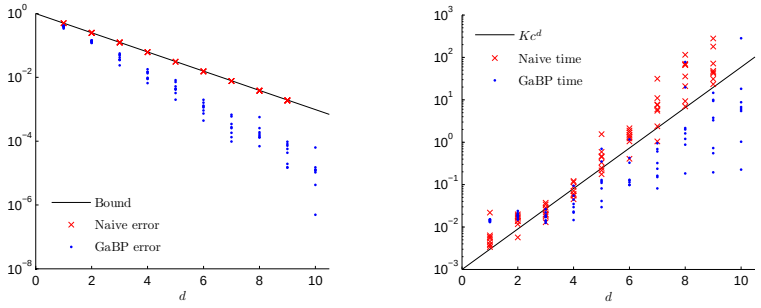
In the case of symmetric matrices with positive paths. It is possible to get good bounds of the expectation E of the error by describing more precisely what paths are in V_{ij}^d. What are the expectations of the number of paths of length $d+1, d+2, ...$ that are or are not in V_{ij}^d ? The paths in V_{ij} are always related to paths in an unwrapped tree rooted in i that end in a replica of j in depth lower then d in the tree. Hence, all the paths without U-turn of size bigger then d are not in V_{ij}^d. Using this idea we can for instance get

$$\frac{\left(\frac{c\gamma}{c-1}\right)^{-d}}{1 - \left(\frac{c\gamma}{c-1}\right)^{-1}} \leq E$$

3.2 Comparison with the naive algorithm

Now, we compare the precision and the complexity of GaBP with the naive algorithm. The difference of precision between the two algorithm is related to the third overestimation of our bound presented in the previous section. Hence, to the $(i, j) \in V^2$ such that $A_{ij} \neq 0$ and $A_{ji} \neq 0$. The more such nodes we have, the more our algorithm is better then the naive. It is not surprising since not null messages move only between such nodes. The two extrema cases are when we do not have such nodes, then the first step of the algorithm is useless since all the messages that circulate are null. The GaBP algorithm corresponds then exactly to the naive. The other extrema case is the symmetric matrix. Not null message travel in every edge and the improvement will be the biggest. Whereas the bound is the same, the expectation of the error of GaBP algorithm is always smaller.

As we already said early in the complexity section, the difference in complexity between the two algorithm is due to the same nodes $i \neq j$ and the analysis is then the same. We observe this differences between both algorithms in figure 6.



(a) Error (b) Complexity

Figure 6: Comparison with the naive algorithm for symmetric ER matrices with positive paths.

Conclusion

I saw during this internship how to invert matrices with the Belief Propagation algorithm. We proved that for diagonally dominant matrices the algorithm converges and we bound the error. We get better approximation and better complexity then the naive algorithm in particular when the matrices are symmetric. As a future work, we may search how it behaves with weaker condition then diagonally dominance. Conversely we could see if we can get better result for more restrictive classes of matrices, for instance matrices with few loops.

Acknowledgments

I thank Devavrat Shah and all his team who allow me to discover the world of research and guided me in my work. I thank also my office mate Romain Hollanders who made my visit at MIT a terrific experience and my days there more pleasant.

References

[1] Danny Bickson. Gaussian Belief Propagation : Theory and Application. 2009.

[2] J. S. Yedidia, W. T. Freemann and Y. Weiss. Understanding Belief Propagation and its Generalizations. 2002.

[3] Y. Weiss and W. T. Freemann. Correctness of Belief Propagation in Gaussian Graphical Models of Arbitrary Topology. 2001.

[4] D. M. Malioutov, J. K. Johnson, A. K. Willsky, Walk-Sums and Belief Propagation in Gaussian Graphical Models. *Journal of Machine Learning Research 7*, 2006.

Annexe

GaBP.m

```
function C = GaBP(A,d)

[x,n] = size(A);

% We save the neighbors and the degrees of the nodes to avoid redundant
% calculation. N is such that N(i,1),...,N(i,deg(i)) are the neighbors of i
% and N(i,k)=0 for k>deg(i).

N = zeros(n);
degree = zeros(1,n);
for i=1:n
    Ni = find(A(i,:));
    Ni = Ni(Ni~=i);
    di = size(Ni);
    degree(i)  = di(2);
    N(i,:) = [Ni zeros(1,n-degree(i))];
end

% Step 1 : d iterations of the message passing rules.

P = diag(diag(A));
p = diag(A); % sum of messages coming to a node.
P1 = P; % new generation of messages without erasing the last one.
for l=1:d
    for i=1:n
        for j=1:degree(i)
            P1(i,N(i,j)) = -A(i,N(i,j))*A(N(i,j),i)/(p(i)-P(N(i,j),i));
        end
    end
    % We can now uptade the messages P and the vector p.
    P = P1;
    for i=1:n
        p(i) = P(i,i);
        for k=1:degree(i)
            p(i) = p(i) + P(N(i,k),i);
        end
    end
end
```

```
% Step 2 : Bd, a first approximation of the inverse . Only the
% diagonal values and the neighbors values B(i,j) are filled. We use the
% equation2 (1) and (2).

B = zeros(n);
for i=1:n
    B(i,i) = 1/p(i);
    for j=1:n
        if (j~=i)
            Q = [0 A(i,j) ; A(j,i) 0];
            Q(1,1) = p(i)-P(j,i);
            Q(2,2) = p(j)-P(i,j);
            Q = inv(Q);
            B(i,j) = Q(1,2);
            B(j,i) = Q(2,1);
        end
    end
end

% Step 3 : local corrections with DFS.

    function C = DFS(C,weight,i,previousNode,j,depth)
        if (depth<d)
            for k=1:degree(j)
                if (N(j,k)~=previousNode) % No U-turn
                    %The path : i-...-previousNode-j-N(j,k)
                    newWeight = weight*B(j,N(j,k))/B(j,j);
                    C(i,N(j,k)) = C(i,N(j,k)) + newWeight;
                    C = DFS(C,newWeight,i,j,N(j,k),depth+1);
                end
            end
        end
    end

C = diag(diag(B));
for i=1:n
    C = DFS(C,B(i,i),i,i,i,0);
end

end
```