Linear-Time Inverse Covariance Matrix Estimation in Gaussian Processes

Joseph Gonzalez
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
jegonzal@cs.cmu.edu

Sue Ann Hong
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
sahong@cs.cmu.edu

Abstract

The computational cost of Gaussian process regression grows cubically with respect to the number of variables due to the inversion of the covariance matrix, which is impractical for data sets with more than a few thousand nodes. Furthermore, Gaussian processes lack the ability to represent conditional independence assertions between variables. We describe iterative proportional scaling for directly estimating the precision matrix without inverting the covariance matrix, given an undirected graph and a covariance function or data. We introduce a variant of the Shafer-Shenoy algorithm combined with IPS that runs in $O(nC^3)$-time, where C is the largest clique size in the induced junction tree. We present results on synthetic data and temperature prediction in a real sensor network.

1 Introduction

Gaussian processes are used in regression and classification tasks in a wide variety of applications including sensor networks, computer vision, localization, relational learning, and control systems. They provide a unified probabilistic framework for modeling the distribution over unobserved values given observed values in a continuous domain. Unfortunately, the use of Gaussian processes in many settings is limited by the cubic time[1] complexity of inference. Furthermore, there is no natural mechanism for Gaussian processes to encode conditional independence between variables. In this paper we will treat marginals of a Gaussian process as a sparse Gaussian graphical model and apply a variant of iterative proportional scaling to obtain a linear time[2] implementation of Gaussian process regression.

1.1 Gaussian Processes

The Gaussian process is a nonparametric generalization of the joint normal distribution defining the distribution over an infinite set of variables. The joint normal distribution over n variables is defined by $Y \sim \mathcal{N}(\mu \in R^n, \Sigma \in R^{n \times n})$ where μ_i is a vector of means indexed by integers $i \in [1, n]$ and $\Sigma_i j$ is a covariance matrix indexed by integers $i, j \in [1, n]$.

[1] Cubic in the number of observations.

[2] Linear in the number of observations assuming ubiquitous conditional independence.

If we instead allow $i, j \in R^d$ then we can define an analogous mean function $\mu(i)$ and covariance function $\Sigma(i, j)$. For example, $i, j \in R^2$ could correspond to locations in a room and $Y_i = Y(i)$ and $Y_j = Y(j)$ could correspond to the temperature random variables at each location. Then the distribution over the infinite random vector Y becomes a distribution over a random *function* defined by the Gaussian process $Y(\cdot) \sim \mathcal{GP}(\mu(\cdot), \Sigma(\cdot, \cdot))$.

Learning a Gaussian process prior consists of learning the $\mu(\cdot)$ and $\Sigma(\cdot)$ functions from the data. The mean function is typically chosen to be the constant zero vector and the covariance function is typically parametric and optimized for the training data. For a set of observations[3] $\{(x_i, y_i)\}_{i=1}^n$ and a *set*[4] of test points $\{x_i^*\}_{i=1}^m$ we define a random vector $Y_z = (Y_{x_1}, \ldots, Y_{x_n}, Y_{x_1^*}, \ldots, Y_{x_m^*})$ and an index vector $z = (x_1, \ldots, x_n, x_1^*, \ldots, x_m^*)$. If we define the distribution over the function $Y(\cdot)$ by $Y \sim \mathcal{GP}(\mu(\cdot), \Sigma(\cdot, \cdot))$, then the marginal distribution over Y_z, the observations and test points is given by Equation (1) and the posterior conditional distribution over the the test points given the observations is given by Equation (2).

$$Y_z \sim \mathcal{N}\left(\begin{bmatrix} \mu(x) \\ \mu(x^*) \end{bmatrix}, \begin{bmatrix} \Sigma(x,x) & \Sigma(x,x^*) \\ \Sigma(x^*,x) & \Sigma(x^*,x^*) \end{bmatrix}\right) \tag{1}$$

$$\begin{aligned} Y_{x^*}|Y_x = y_x \sim\ & \mathcal{N}[\mu(x^*)\Sigma(x^*,x)\Sigma(x,x)^{-1}(y_x - \mu(x)), \\ & \Sigma(x^*,x^*) - \Sigma(x^*,x)\Sigma(x,x)^{-1}\Sigma(x,x^*)] \end{aligned} \tag{2}$$

Note that in Equation (2) we must invert the matrix $\Sigma(x, x)$. For most kernels this matrix will be dense and the computational complexity of inverting an $n \times n$ matrix is approximately[5] $O(n^3)$. In cases where there are many observations and n is large, matrix inversion may be intractable. In fact, in some settings even writing down the Σ matrix may be too costly. In the next section we introduce the Gaussian graphical model and show how it can compactly represent a marginal of a Gaussian process. Using this compact representation we will be able to efficiently compute the posterior distribution given in Equation (2).

1.2 Gaussian Graphical Models

A Gaussian graphical model is a compact representation of a joint normal distribution. Given an undirected graph $G = (V, E)$, the joint probability distribution of a Gaussian graphical model can be written in the form of Equation (3). The factors of the model correspond to the edges and vertices in G.

$$P(Y|\Psi) = \frac{1}{Z} \prod_{(i,j)\in E} \exp[-\psi_{ij}(Y_i, Y_j)] \prod_{i \in V} \exp[-\psi_i(Y_i,)] \tag{3}$$

The explicit form of the clique potentials ψ_{ij} and ψ_i are given in Equation (4). The η term, typically used in the natural parameterization of the joint normal, is defined as $\eta = \Lambda\mu$.

$$\psi_{ij}(Y_i, Y_j) = \frac{1}{2} Y_i Y_j \Lambda_{ij}, \qquad \psi_i(Y_i) = \frac{1}{2} Y_i^2 \Lambda_{ii} - Y_i \eta_i \tag{4}$$

By replacing the clique potentials in equation (3) with equation (4) and taking the sum of the exponents we obtain equation (5).

$$\begin{aligned} P(Y|\Psi) &= \frac{1}{Z} \exp\left[-\frac{1}{2}\left(\sum_{(i,j)\in E} Y_i Y_j \Lambda_{ij} + \sum_{i \in V} Y_i^2 \Lambda_{ii} - \sum_{i \in V} Y_i \eta_i\right)\right] \\ &= \frac{1}{Z} \exp\left[-\frac{1}{2}\left(Y^T \Lambda Y - Y^T \eta\right)\right] \end{aligned} \tag{5}$$

[3] An observation is a location-measurement pair.

[4] Several simultaneous dependent predictions can be made using a Gaussian process.

[5] Current best asymptotic complexity is $O(d^{2.376})$

From Equation (5) it is clear that a Gaussian graphical model represents a joint normal distribution. More formally, for a particular graph G the corresponding Gaussian graphical model represents the set of normal distributions given in Equation (6) where $\Lambda = \Sigma^{-1}$ is a positive definite matrix.

$$N(G) = \{N(\mu, \Sigma) \mid (\Sigma^{-1})_{ij} = \Lambda_{ij} = 0 \iff (i,j) \notin E\} \tag{6}$$

For typical kernel functions, the corresponding Gaussian graphical model would be fully connected as in Figure 1(a). However, in certain domains we may believe that a sparse graph such as Figure 1(b) would more accurately capture the independence structure.

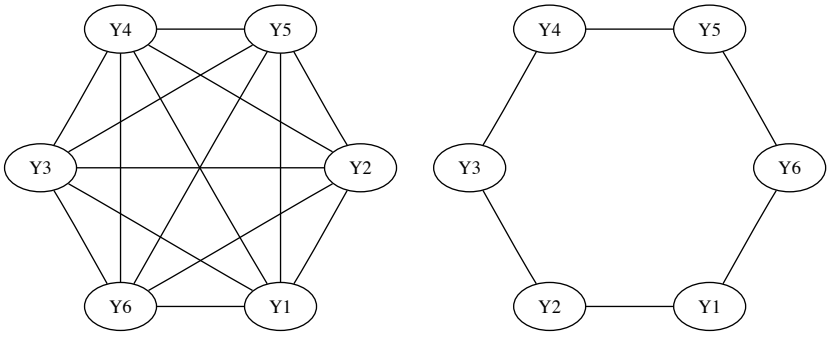


Figure 1: Both graphs represent a Gaussian graphical model over 6 variables. The graph in (a) is a fully connected Gaussian graphical model which corresponds to the marginal obtained from a Gaussian process using most kernels. The graph in (b) is a partially connected Gaussian graphical model which would have a sparse precision matrix.

Next we will show how, given a relatively sparse graph for the set of variables in the marginal distribution and a kernel function, we can efficiently and provably learn the optimal set of parameters for the corresponding Gaussian graphical model.

2 Algorithm

2.1 Generalized Iterative Scaling for Gaussian Graphical Models

Generalized iterative scaling (GIS) is an iterative algorithm commonly used when no closed form for the maximum likelihood estimate (MLE) of parameters exists. As such, it is the standard method for finding parameters in undirected graphical models. Here we illustrate a generalized iterative scaling algorithm for estimating Λ in Gaussian graphical models. The derivation is similar to [3] and [4].

Let us first consider the case where we have an *iid* sample $D = Y^1, \ldots, Y^m; Y^i \in R^n$. Assume the mean vector $\mu = \mathbf{0}$ (we can generalize to non-zero mean cases easily by working in the zero-mean space and translating results to the sample mean). Then the explicit form of the joint normal pdf for D can be written as

$$p(D|\Lambda) = \frac{|\Lambda|^{nm/2}}{\sqrt{2\pi}^{nm}} \exp\left[-\frac{1}{2}\sum_{l=1}^{m}(Y^l)^T \Lambda (Y^l)\right] = \frac{|\Lambda|^{nm/2}}{\sqrt{2\pi}^{nm}} \exp\left[-\frac{1}{2} tr\{\Lambda W\}\right] \tag{7}$$

where $W = \sum_{l=1}^{m} Y^l (Y^l)^T$. Note that W has a Wishart distribution $W_d(n, \Sigma)$, and for $m \geq n$, W is positive definite for positive definite Σ. However the maximum likelihood

estimator of Σ is given by Equation (8). Thus we can use the kernel function wherever we would use the Wishart matrix.

$$\Sigma(\cdot,\cdot) = \hat{\Sigma} = \frac{1}{m}\sum_{l=1}^{m}(Y^l - \mu)(Y^l - \mu)^T = \frac{1}{m}\sum_{l=1}^{m}(Y^l)(Y^l)^T = \frac{W}{m} \tag{8}$$

The iterative update equation for GIS is designed to update submatrices of Λ to satisfy Equation (8) and is given for each factor f by

$$\Lambda_{ff}^{(t+1)} = \Sigma(f,f)^{-1} + \Lambda_{fa}^{(t)}(\Lambda_{aa}^{(t)})^{-1}\Lambda_{af}^{(t)} \tag{9}$$

where $a = V \backslash f$. We can see that a fixed point $\Lambda = \hat{\Lambda}$ satisfies Equation (8) as

$$\hat{\Sigma}_{ff} = (\hat{\Lambda}^{-1})_{ff} = (\hat{\Lambda}_{ff} - \hat{\Lambda}_{fa}(\hat{\Lambda}_{aa})^{-1}\hat{\Lambda}_{af})^{-1} = (\Sigma(f,f)^{-1})^{-1} = \Sigma(f,f)$$

Starting with $\Lambda^{(0)} = I_d$, the IPS algorithm iteratively applies equation (9) to every factor f until convergence, where each time step entails sequentially updating all submatrices of Λ corresponding to the factors.

2.2 Properties of the IPS Update Equation

The update equation satisfies two desirable properties. First, if $\Lambda^{(t)}$ is positive definite, then $\Lambda^{(t+1)}$ is also positive definite [4], since $\Sigma(f,f)^{-1}$ is positive definite. Second, the precision remains zero where there are no edges. Also, each update is monotonic increasing in the likelihood of the data. Because the update operator is continuous and the likelihood function is compact and concave in the parameter space, IPS is guaranteed to eventually converge to the MLE [1]. Although in general the number of iterations is not bounded, in practice IPS tends to converge after a small number of iterations.

For chordal graphs, a closed-form solution for MLE of Λ exists that requires a single calibration of the junction tree. The time complexity of solving the closed-form solution is thus $O(nC^3)$ where C is the size of the largest clique in the junction tree. As we will see, one iteration of our IPS algorithm as described in Section 2.4 has the same time complexity and will converge for chordal graphs. However, there is no closed-form solution for non-chordal graphs. Since transforming a non-chordal graph to a chordal graph introduces unwanted edges and thus dependencies between variables, learning the parameters via IPS for the exact graph structure may be beneficial.

2.3 Junction Tree

As shown in Equation (9), the IPS update requires the calculation of $\Lambda_{fa}^{(t)}(\Lambda_{aa}^{(t)})^{-1}\Lambda_{af}^{(t)}$. As our factors are of size 1 or 2, $(\Lambda_{aa}^{(t)})^{-1}$ means inverting a $(n-1)$-by-$(n-1)$ or a $(n-2)$-by-$(n-2)$ matrix, which is again $O(n^3)$ in time (we will use factors and precision Λ_{ff} interchangeably since each factor is fully characterized by the corresponding submatrix of Λ). Applying the definition of the marginal Λ_{ff}, we can rewrite the update equation as

$$\Lambda_{ff}^{(t+1)} = \Sigma(f,f)^{-1} + \Lambda_{fa}^{(t)}(\Lambda_{aa}^{(t)})^{-1}\Lambda_{af}^{(t)} = \Sigma(f,f)^{-1} + \Lambda_{ff}^{(t)} - \Lambda_{ff}^{m(t)} \tag{10}$$

where $\Lambda_{ff}^{m(t)}$ represents the precision for the marginal distribution of f according to $\Lambda^{(t)}$, since marginalization of normal distributions give us $\Lambda_{ff}^{m(t)} = \Lambda_{ff}^{(t)} - \Lambda_{fa}^{(t)}(\Lambda_{aa}^{(t)})^{-1}\Lambda_{af}^{(t)}$.

To compute the marginals $\Lambda_{ff}^{m(t)}$ we use junction tree calibration, a technique used in graphical models for *exact* inference. In Gaussian graphical models we can calibrate the junction tree in $O(nC^3)$-time, where C is the size of the largest clique. To apply the junction

tree algorithm we first triangulate the graph and then define the vertices to be maximal cliques in the resulting chordal graph. Finally we construct a junction tree from the chordal graph, on which we can apply Shafer-Shenoy to get a calibrated tree. However, performing calibration after every factor IPS update is expensive. We describe a faster algorithm in Section 2.4 that allows us to perform only one instance of calibration while updating every factor at least once.

We expect our algorithm to run much faster than matrix inversion for large n. However, creating a junction tree incurs additional costs. Running the min-fill heuristic efficiently requires maintaining a sorted list which can be done in $O(nd \log n)$ time, where d is the maximum degree of the graph. Alternatively we could use an arbitrary heuristic which runs in linear time. Creating a junction tree from the induced chordal graph is known to be possible in $O(n + m)$-time, where m is the number of edges in the chordal graph [2]. However, we are considering only sparse graphs, in which cases finding a junction tree is achievable in linear time. In Section 3.3 we present experimental results demonstrating that our algorithm runs in near linear time.

2.4 Computing the Update Equation: Fast Exact Inference via Junction Tree

Naively applying junction tree, we must re-calibrate the junction tree after every factor update, which leads to $O(n^2C^3)$-time complexity per IPS update, defined as updating all factors (at least) once. However, by by cleverly updating factors we are able to run the entire junction tree calibration only once per IPS update.

Two insights lead to our $O(nC^3)$-time IPS update algorithm. First, we can update factors associated with a clique without affecting the beliefs of other cliques. Recall that at calibration the marginal probability of clique i is characterized by the initial belief π_i^o and the inbound messages δ to i; here the marginal precision is $\Lambda_i = \pi_i^o + \sum_{j \in Neighbor(i)} \delta_{ji}$. By updating π_i^o to reflect the new value of Λ we maintain a calibrated clique. This leads to Algorithm 1 for updating all factors in a clique sequentially.

Algorithm 1: UpdateFactors($\mathcal{T}$,i)

input : Clique tree $\mathcal{T} = (\Pi = \{\pi_i^o \text{ for each clique } i\}, \Delta = \{ \text{ messages } \delta_{ij}\}, \text{Children})$, Current node i

output: Updated clique tree $\mathcal{T}'$ with updated π_i^o representing the updated Λ

begin

  for *factor f represented in clique i* **do**

    Compute the marginal precision for the factor f :

    $\Lambda_{ff}^{m(t)} = \text{marginalize\_out}(\pi_c^o + \sum_{j \in Neighbor(c)} \delta_{jc}, c \backslash f)$

    Get the new factor value for Λ : $\Lambda_{ff-tmp} = (\Sigma_{ff})^{-1} + \Lambda_{ff} - \Lambda_{ff}^{m(t)}$

    Update the initial belief : $(\pi_i^o)_{ff} = \Lambda_{ff-tmp} + (\pi_i^o)_{ff} - \Lambda_{ff}$

end

Second, if we update any clique i using Algorithm 1, then the child j of i will be calibrated upon receiving a new message from i. Because no other initial belief changes, messages from other cliques into j will remain the same. Thus upon receiving the message for i, the clique j is calibrated. Applying this intuition, algorithm 2 describes an algorithm that traverses the junction tree in depth first search order, updating each clique and sending messages from the current clique to the next. The algorithm can be viewed as a Gaussian, DFS version of the efficient IPF update scheduling discussed in [5] for general graphical models.

Algorithm 2: DFS-jtIPS($\mathcal{T}$,i)

input : Clique tree $\mathcal{T} = (\Pi = \{\pi_i^o \text{ for each clique } i\}, \Delta = \{ \text{ messages } \delta_{ij}\}, \text{Children})$, Current node i

output: Updated Clique Tree T' with updated factors (Λ) for the subtree rooted at i and calibrated message from i to its parent

begin

  T' = UpdateFactors($\mathcal{T}$,i) ;

  for *child c in Children(i)* **do**

    Send message from i to child c :

    $\delta_{ic} = \text{marginalize\_out}(\pi_i^o + \sum_{j \in Neighbor(i) \setminus c} \delta_{ji}, \text{Separator}(i, c))$;

    Update the subtree rooted at the child c : $\mathcal{T}'$ = DFS-jtIPS($\mathcal{T}'$,c) ;

  if *i has a parent j* **then**

    Send message from i to parent j :

    $\delta_{ij} = \text{marginalize\_out}(\pi_i^o + \sum_{k \in Neighbor(i) \setminus j} \delta_{ki}, \text{Separator}(i, j))$;

end

3 Experimental Results

3.1 Synthetic Data

Synthetic data was used to precisely compare the distributions obtained from the estimated precision matrix to the true distributions. First we constructed a random connected graph and used the kernel function $\Sigma_{ij} = \exp(-\frac{1}{2}(i-j)^2)$ to create an initial covariance matrix. After inverting Σ to get Λ, we zeroed out entries in Λ corresponding to no edges. To increase the chances of obtaining a positive semidefinite matrix we reinforced the diagonal of Λ with an identity matrix since we only used the resulting Λ if it was positive semidefinite. We used the true inverse of Λ as the sample covariance to test that our algorithm converges to the true value of Λ.

3.2 KL Divergence

We used KL-divergence of the estimated $\tilde{\Lambda}$ to the true value of Λ as the error metric. The expression for the KL divergence over the normal distribution is given by Equation (11).

$$\text{KL}(\Lambda||\tilde{\Lambda}) = \frac{1}{2}\log\left(\frac{|\Lambda|}{|\tilde{\Lambda}|}\right) + \frac{1}{2}\text{Tr}\left(\tilde{\Lambda}\Lambda^{-1}\right) - \frac{d}{2} \tag{11}$$

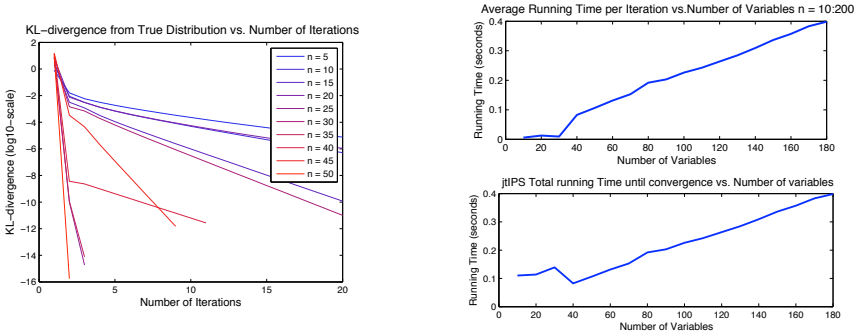
It is important to note that KL divergence is not symmetric. We will always compute the KL divergence in the direction $\text{KL}(\Lambda||\tilde{\Lambda})$.

3.3 Algorithm Performance

Figure 2(a) shows the performance of our algorithm for different numbers of variables, measured by KL-divergence over iterations. The algorithm converges quickly, taking only a few iterations in most cases. In Figure 2(b) we compare the running time of the algorithm with respect to the number of variables n, averaged over 20 trials for each number of variables. We observe that the running time increases linearly as a function of n.

3.4 Berkeley Sensor Network

We tested our algorithm on Berkeley sensor networks data. We believe that the building floor-plan imposes conditional independence constraints on the sensors. For example, sen-

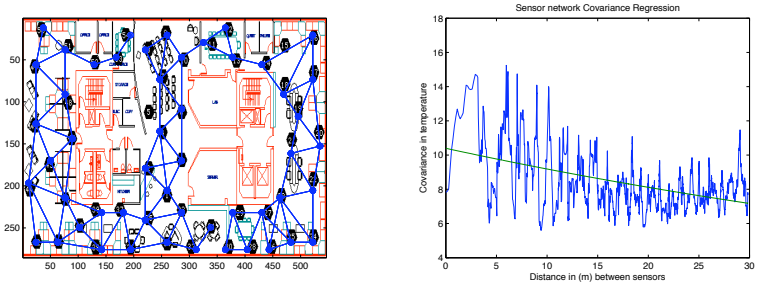


(a) (b)

Figure 2: In (a) the KL divergence is plotted as a function of the number of iterations. Each line in (a) corresponds to a different number of variables n. In (b) the running time of IPS per iteration and per convergence are shown.

sors at opposite ends of the hallway should be conditionally independent given the sensors between them.

The Berkeley sensor networks data set[6] contains the coordinates of 54 sensors, as well as time series data for the temperature, light, humidity, and voltage level of the batteries for each sensor. Here we considered only the temperature data. Given the layout of the sensors and spatial characteristics of the room such as walls and doors, we manually constructed the graph shown in Figure 3(a) with about 100 edges. The data required preprocessing due to missing entries and malfunctioning sensors. We removed 2 sensors and several samples from the time series which were completely empty. The last two days (weekend) of data were removed due to inconsistent behavior. Linear interpolation was used to fill in missing entries. The mean was subtracted from all data points to produce a zero-mean data set. The final data set consisted of 52 sensors each with 5000 recordings.



(a) (b)

Figure 3: (a) Layout of the Berkeley lab with manually drawn edges in blue. Notice that sensor 5 and 15 were removed due to low reporting. (b) Covariance vs distance and the fit exponential kernel.

The data were randomly divided into a training set of 4000 time steps and a test set of 1000. We fit a simple exponential kernel of the form in Equation (12) to the sample covariance of

[6] http://www.cs.cmu.edu/ guestrin/Research/Data/

the training set as seen in Figure 3(b).

$$K(x_1, x_2) = 10.3957 \exp(-0.012377||x_1 - x_2||_2) \tag{12}$$

Using the learned kernel, sample covariance from the training set, and the IPS algorithm, posterior predictions were made for each time step in the test set by removing a random sensor and then predicting its value. The resulting root mean squared errors are shown in Table 1. Additionally the KL divergence between the sample covariance, the kernel only, and the kernel with graphical model are also shown in Table 1. The KL divergence between the kernel-only GP and GP with graphical model $KL(\Sigma(\cdot,\cdot)||\Sigma_{graph})$ was 4.1262.

	$\hat{\Sigma}$	$\Sigma(\cdot,\cdot)$	Σ_{graph}	Identity Matrix
RMSE	0.1676	0.6934	0.7343	3.2469
KL($\hat{\Sigma}||\cdot$)	0	60.1332	64.1575	318.7162

Table 1: Comparison of the root mean squared error in the prediction of the test data (random missing sensor at each time step) using different covariance matrices. $\hat{\Sigma}$ is the sample covariance, $\Sigma(\cdot,\cdot)$ is the covariance from the estimated kernel, and Σ_{graph} is the covariance learned from the kernel and the graph via IPS. The identity matrix is used as a baseline. KL-divergence to the sample covariance is also shown.

We had expected combining the graphical representation with the kernel to yield a lower root mean squared error than using the kernel by itself. However, from Table 1 we can see that adding the graphical interpretation gave slightly worse predictions than using just the kernel function. One explanation may be that the graph does not accurately reflect the conditional independence structure of the room. For example, all sensors near windows were linked by the outside temperature and therefore not conditionally independent even though the floor plan does not suggest strong spatial linkage between them.

4 Conclusions

We presented an algorithm that enables Gaussian processes regression in $O(nC^3)$-time given a graph. Although our algorithm was able to retrieve the true precision matrices for synthetic data, it did not outperform GP regression without a graph on the experiment for real sensor network. However, we believe that with more principled methods for kernel hyperparameter training and constructing the graph, our algorithm can be beneficial to a large array of domains.

References

[1] M. Eichler. Gaussian graphical models. http://statlab.uni-heidelberg.de/people/eichler/graphs/graph3.pdf.

[2] Philippe Galinier, Michel Habib, and Christophe Paul. Chordal graphs and their clique graphs. In *Workshop on Graph-Theoretic Concepts in Computer Science*, pages 358–371, 1995.

[3] M. I. Jordan. *An Introduction to Probabilistic Graphical Models*. in preparation.

[4] S. Lauritzen. Log-linear and gaussian graphical models, saint flour summerschool lectures. http://www.stats.ox.ac.uk/ steffen/stflour/sf5c.pdf, 2006.

[5] Y. Teh and M. Welling. On improving the efficiency of the iterative proportional fitting procedure. In *Proc. of the 9th Int'l. Workshop on AI and Statistics (AISTATS-03)*, 2003.