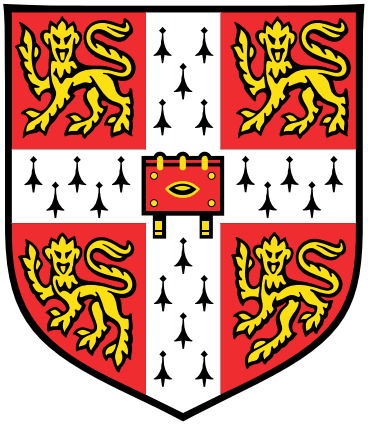
Distributed Gaussian Process Regression in Networked Systems

Aman Sinha
Churchill College
University of Cambridge



Submitted to the
Department of Engineering
for the degree of
Master of Philosophy

August 2014

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text.

Acknowledgements

I would like to thank numerous individuals who helped me put forth my best efforts in this study. First of all, I recognize my supervisor, Glenn Vinnicombe, and my advisor, Carl Rasmussen, for fruitful discussions and constructive suggestions regarding my work. I would also like to thank Max Rabinovich, Tiras Lin, and many other colleagues for their feedback and camaraderie. Finally, I would like to thank the Winston Churchill Foundation of the United States for funding my research and for providing me the opportunity to spend this year in Europe. It was certainly an unforgettable experience.

This thesis is dedicated to my family, for their unconditional love, support, and guidance.

Abstract

Performing nonparametric regression in networked systems presents fundamental challenges above and beyond the development of scalable and distributed protocols. Our work develops protocols that are indeed scalable and distributed by design but are focused on addressing the more foundational concerns of accuracy and robustness. We measure accuracy in terms of the ability of our regression network to perform competitively with centralized approaches to nonparametric regression, and we measure robustness in terms of the network's ability to resist the impact of data loss via nodal failure.

Building upon the Gaussian process (GP) framework of Bayesian nonparametric regression, we develop the Bayesian Committee Machine of Experts (BCME), a sparse GP model that combines the efficiency of popular sparse GPs with the modeling flexibility of mixture-of-experts models. Empirical results indicate that our method performs very competitively with centralized sparse GPs including the FITC approximation[20] and the variational sparse GP[23]: the BCME is capable of providing the same errors on test datasets as these centralized approaches but often with significantly faster training times.

To improve robustness to nodal failure, we develop a moment-matching protocol that provides a principled way for nodes to exchange data with each other. In addition, we prove bounds on the number of data exchanges that need to occur to guarantee a certain degree of robustness. Our protocol is based on an existing asynchronous gossip algorithm[2]; the novelty of our approach is the improvements we add to render our protocol both adaptable to realistic network scenarios and better suited to working in tandem with the BCME. Our sophistications result in significant reductions in computation time compared to the original algorithm, as measured over a variety of network configurations.

The beauty of our method is its modularity. By tackling accuracy and robustness separately, we have created an approach that is flexible and easily tunable to many applications. Nevertheless, our analysis has revealed tradeoffs that arise between robustness and accuracy, resulting in an implicit Pareto front between the two objectives. We propose numerous improvements for both protocols to diminish the magnitudes of these tradeoffs in future research. Together, the BCME and the moment-matching protocol represent the foundations of a larger framework for scalable, distributed nonparametric regression.

Contents

List of Figures

Chapter 1

Introduction

In the supervised learning framework of machine learning, the task is to infer a latent function given (potentially corrupted or noisy) labelled examples. The goal of supervised learning algorithms is to then use this learnt function to make predictions at new input locations. When the inputs and outputs are real-valued data, this task is simply that of performing regression. In this way, there are innumerable applications for supervised learning models across many fields of inquiry. Indeed, whether performing high-frequency trading in financial markets or predicting weather patterns using an off-shore sensor network, performing regression accurately and efficiently already lies at the core of many aspects of our daily lives. These applications will only increase as the sophistication of automated regression techniques improves, since systems that can perform accurate and efficient regression have the potential to learn and adapt their behaviors to changing environments.

Probabilistic methods of inference, and more specifically Bayesian nonparametric methods, have received a considerable amount of attention in supervised learning. These techniques provide a principled approach to learning functions that is appealing for two main reasons. First, by placing a prior distribution over all possible functions, these models are flexible enough to model arbitrarily complex data, as opposed to parametric models which are inherently limited. Second, these models have the capability of performing model selection, a process that automatically balances model complexity with data fit and therefore attempts to discover the underlying structure of the data. In this thesis, we will focus on a particular Bayesian nonparametric method known as the Gaussian Process (GP) model, which places a Gaussian process prior over functions. Although GPs have been used in meteorology and geostatistics since the mid-20th century, machine learning researchers have only been studying GPs for roughly the past two decades, as evidenced by the relatively recent publication of the standard text on GP models by Rasmussen and Williams [15].

We will focus on the task of performing GP regression in a network setting. As datasets grow in size and data collection becomes increasingly distributed, it is often necessary to perform regression in a distributed, or even more stringently, a decentralized manner. In fact, such scenarios often arise very naturally in various application scenarios. For example, in the off-shore wireless network mentioned above, each sensor must collect data and communicate with the other sensors (or perhaps a centralized base) to make predictions. Another scenario is that of a fleet of autonomous vehicles, wherein each vehicle learns about its local environment and communicates this knowledge with the others in order for the group to make an informed decision on the direction of travel.

1.1 Objectives and Approach

In the context of decentralized systems, scalability and efficiency are of utmost importance, but they are not the only concerns. Rather, distributed protocols must also be judged in terms of their accuracy and robustness. Of course, the precise definitions of accuracy and robustness can vary across applications. We will determine accuracy of our distributed GP model by how well the system can perform predictions compared to centralized approaches in which all of the data is globally accessible (as measured, for example, by the mean square error or log probability metrics on a test set of data). When each node is collecting its own data, the failure of a node results in a loss of data. We will define robustness against nodal failure as the resistance of a system's predictive power against such losses of data; our methods will build upon the intuition that a system with a given number of nodes is most robust when data is randomly distributed. Our goal is to develop methods that can perform GP regression accurately and robustly in a scalable, efficient, and distributed manner.

This objective is the amalgamation of two disparate but inherently related tasks. We will observe a subtle interplay between the competing goals of accuracy and robustness to nodal failure. Using our proposed protocols, we will illustrate an implied Pareto frontier, wherein increasing robustness comes at the expense of accuracy and vice versa. Our analyses of each objective will also be quite different from each other. Our analysis of accuracy combines two popular research themes in GP regression techniques: sparse models and mixture-of-experts models. In contrast, our methods for tackling the robustness problem draw upon techniques and methodologies of control theory. In particular, we employ standard techniques of consensus dynamics in a novel way.

The remainder of this report is organized as follows. Chapter 2 presents a sparse, decentralized GP regression model we denote as the Bayesian Committee Machine of Experts (BCME). We evaluate this model in terms of its accuracy and efficiency compared to commonly used centralized GP models. Chapter 3 presents an algorithm for increasing robustness to nodal failure by requiring nodes to exchange random subsets of data. We discover bounds on the number of exchanges that must occur to reach a certain level of robustness. Because our two research themes and methodologies are quite different, we present relevant background material and notation within each chapter. The study is concluded with a discussion of the results, their implications, and open questions for further research.

Chapter 2

The Bayesian Committee Machine of Experts

Motivation

Consider the task of accurately performing predictions in a distributed network while maintaining scalability. In this context, we are primarily concerned with two characteristics of our protocol: the local predictions performed by nodes with their local datasets and the technique by which we combine these local predictions to produce a more accurate global prediction. To guarantee flexibility and simplicity, we will build our inference protocol in a Bayesian nonparametric manner. The Gaussian process (GP) model is a powerful technique to perform Bayesian nonparametric regression and classification. As a fully probabilistic Bayesian technique, it reports predictions with associated uncertainties, which can be particularly important in applications related to controlling system behavior. Furthermore, as with other nonparametric methods, this tool is flexible enough to model complex and nonlinear datasets in exactly the same way as it models simple datasets. Our challenge is to design a technique which can accurately perform GP regression in a decentralized, scalable manner.

This objective subsumes two popular research trends on GP regression techniques. The first concerns sparse methods. Standard GP regression has $O(N^3)$ training time complexity and $O(N^2)$ prediction time complexity per test case, respectively, for a dataset with N points. Sparse methods modify the GP regression model to reduce training and prediction time complexities, and most of these methods do so by augmenting the training data with so-called inducing variables. The fundamental assumption is that latent function values interact sparsely via the inducing variables, resulting in exact inference using a modified prior [13]. For example, the fully independent training conditional (FITC) approximation assumes that, given these inducing variables, the latent function values are fully independent (corresponding to a diagonal covariance structure); the partially independent training conditional (PITC) method assumes only partially independence (corresponding to block diagonal covariance structure) [20]. We cover these approximations in more detail below. Importantly, most of the popular sparse approximations do not concern themselves with the ability to perform regression in a decentralized manner (with the exception of the Bayesian Committee Machine [25]).

The second area concerns mixture-of-experts models. These models are often proposed as a means of giving GPs with stationary kernels the extra flexibility of modeling heteroscedastic or

non-stationary data. In the mixture-of-experts approach, the data is partitioned into separate regions, and a separate GP "expert" models the data in each partition [8, 9, 10, 14]. Unfortunately, the partitioning of data is often performed in a centralized manner, requiring intensive MCMC sampling [8, 14] (which also destroys scalability) or the ability to iteratively reassign training data to different experts [9]. Such centralized methods are unacceptable for our model.

Our method will attempt to balance the efficiency of sparse methods with the flexibility of mixture-of-experts approaches in a decentralized, scalable approach. Despite an expected loss in predictive performance, we do not want to perform inference over data partitions. Except for data exchanges that occur to increase robustness to nodal failure (cf. Chapter 3), we assume that the training data on each node is fixed. In this way, we want our model to be able to take advantage of any clustering present in the data but also be able to perform accurately on unclustered data. We will find that there is a subtle interplay between robustness to nodal failure and accuracy of the regression model. Whereas the system is most robust to nodal failure when each node has a random subset of training data, the assumptions of our regression model will be more easily satisfied when the data is clustered. In addition, the system is more robust to nodal failure as the number of nodes increases and each node has less data, but each node then tends to be more susceptible to overfitting and the resulting system is less accurate.

2.1 Background

We now present a brief overview of standard GP regression techniques as well as three popular sparse methods: the fully independent training conditional (FITC) approach, the partially independent training conditional (PITC) approach, and the Bayesian Committee Machine (BCM), which is actually an equivalent, but decentralized reformulation of PITC. Each of these methods will be relevant to our proposed model.

We adopt the notation of $\mathbf{x} \in \mathbb{R}^d$ as a d-dimensional input data point and y as a target. A dataset consisting of N d-dimensional input points will be denoted by the input matrix X with a corresponding target vector $\mathbf{y}$.

2.1.1 Gaussian Processes for Regression

The GP model assumes that there is an underlying latent function f at each input point, or equivalently, the underlying vector $\mathbf{f}$ associated with X. The targets are simply noisy observations of these latent values:

$$P(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{f}, \sigma_n^2 I), \tag{2.1}$$

where σ_n denotes a noise level, and $\mathcal{N}(\boldsymbol{\mu}, V)$ is a multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance V. We then assume a GP prior over functions[1]:

$$P(\mathbf{f}|X) = \mathcal{N}(\mathbf{0}, K_{ff}), \tag{2.2}$$

where K_{ff} is the covariance matrix between the latent function values. This matrix is built by the underlying covariance function defining the smoothness properties of the prior: $[K_{ff}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. A popular covariance function that we will employ in our approach is the squared

[1] We will always consider priors with zero mean in this study.

exponential function:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\sum_{a=1}^{d}(x_{i_a} - x_{j_a})^2/l_a^2\right), \tag{2.3}$$

where σ_f defines the hyperparameter for signal magnitude and l_a the characteristic length scale for dimension a. Of course, the overall objective is to determine predictions $\mathbf{f}_*$ at test locations X_* given the training data:

$$P(\mathbf{f}_*|\mathbf{y}, X, X_*) = \int P(\mathbf{f}, \mathbf{f}_*|\mathbf{y}, X, X_*)d\mathbf{f} = \frac{1}{P(\mathbf{y}|X)}\int P(\mathbf{y}|\mathbf{f})P(\mathbf{f}, \mathbf{f}_*|X, X_*)d\mathbf{f}, \tag{2.4}$$

where the joint prior over test and training values is:

$$P(\mathbf{f}, \mathbf{f}_*|X, X_*) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K_{ff} & K_{f*} \\ K_{*f} & K_{**} \end{bmatrix}\right). \tag{2.5}$$

This integral is tractable since all distributions are Gaussian, so we reach the final result:

$$P(\mathbf{f}_*|\mathbf{y}, X, X_*) = \mathcal{N}\left(K_{*f}\left(K_{ff} + \sigma_n^2 I\right)^{-1}\mathbf{y}, K_{**} - K_{*f}\left(K_{ff} + \sigma_n^2 I\right)^{-1}K_{f*}\right). \tag{2.6}$$

The cost of training is $O(N^3)$ to invert the covariance matrix $K_{ff} + \sigma_n^2 I$. Performing precomputations during training time yields prediction times of $O(N)$ for the mean and $O(N^2)$ for the variance per test point.

In Figure 2.1, we show illustrative examples of key aspects for GP models. Figure 2.1(a) shows a simple example of performing GP regression on one-dimensional data. A squared exponential covariance function is used to predict a function nonparametrically given 6 data points. We also show draws from the posterior distribution, which are functions that could have generated the data. Figure 2.1(b) shows the effects of varying hyperaparameters. Functions are drawn from prior distributions with zero mean and covariance matrices given by the hyperparameters shown in the legend. The functions of the hyperparameters are very intuitive: l_1 determines the length scale of correlation, whereas σ_f determines the magnitude of uncertainty/correlation between input points.

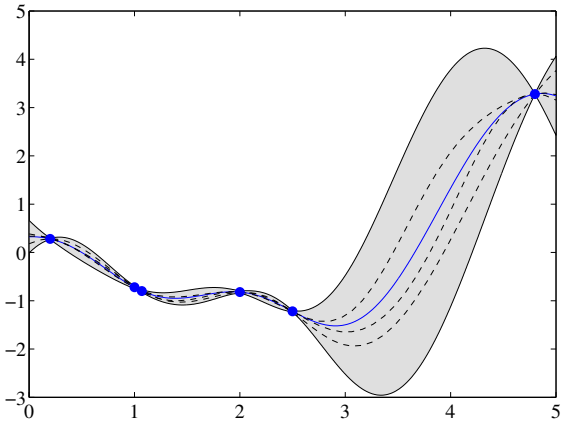
Hyperparameter Training

We will denote $\boldsymbol{\theta}$ as the vector of hyperparameters of the noise magnitude and covariance function. Instead of integrating these variables out, it is much more computationally efficient to form point estimates. This is usually done by optimizing by maximizing the marginal likelihood of the training data $P(\mathbf{y}|X, \boldsymbol{\theta})$. This methodology can easily be derived by first attempting to integrate out $\boldsymbol{\theta}$:

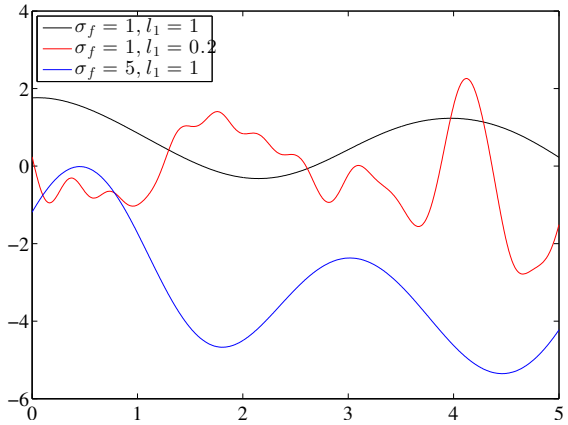
$$\begin{aligned} P(\mathbf{f}_*|\mathbf{y}, X, X_*) &= \int P(\mathbf{f}_*|\mathbf{y}, X, X_*, \boldsymbol{\theta})P(\boldsymbol{\theta}|\mathbf{y}, X)d\boldsymbol{\theta} \\ &= \frac{1}{P(\mathbf{y}|X)}\int P(\mathbf{f}_*|\mathbf{y}, X, X_*, \boldsymbol{\theta})P(\mathbf{y}|X, \boldsymbol{\theta})P(\boldsymbol{\theta}|X)d\boldsymbol{\theta}. \end{aligned} \tag{2.7}$$

We now assume that the marginal likelihood $P(\mathbf{y}|X, \boldsymbol{\theta})$ is peaked at an optimal value $\hat{\boldsymbol{\theta}}$ so that we can approximate it as a delta function. With a flat or vague prior $P(\boldsymbol{\theta}|X)$, we have:

$$P(\mathbf{f}_*|\mathbf{y}, X, X_*) \approx P(\mathbf{f}_*|\mathbf{y}, X, X_*, \hat{\boldsymbol{\theta}}). \tag{2.8}$$



(a) GP regression on one-dimensional data



(b) Functions drawn from priors with different kernel hyperparameters

Figure 2.1. Illustrative examples of GP regression. (a) shows regression on a toy one-dimensional dataset. Data are shown as blue dots, the posterior mean function as a blue line, and the 95% confidence interval of the latent function in gray. Uncertainty grows larger as we attempt to predict in regions devoid of data. The black dotted lines are sample functions drawn from the posterior (Equation 2.6). (b) shows functions drawn from prior distributions with zero mean and covariance matrices given by parameters shown in the legend.

Because we are performing a maximum likelihood estimate on the hyperparameters rather than the actual function values, this type of estimation is termed type II maximum likelihood [15]. The log marginal likelihood is given by the following expression:

$$\log P(\mathbf{y}|X, \boldsymbol{\theta}) = -\frac{1}{2}\log\det\left(K_{ff} + \sigma_n^2 I\right) - \frac{1}{2}\mathbf{y}^T\left(K_{ff} + \sigma_n^2 I\right)^{-1}\mathbf{y} - \frac{N}{2}\log(2\pi) \tag{2.9}$$

The optimization of this function using gradient-based methods requires an $O(N^3)$ matrix inversion at each evaluation of the gradient, so hyperparameter training time represents the largest bottleneck in performing GP regression. This can be ameliorated if we are willing to optimize the hyperparameters on only a subset of data. However, sparse methods using inducing variables represent a more principled approach to reducing both hyperparameter training and regular training time for the GP model.

2.1.2 Sparse Models using Inducing Variables

As mentioned above, many sparse models introduce sparsity through interactions with M inducing variables $\mathbf{u}$ and corresponding inputs X_u. Although the inducing variables are always integrated out, the locations X_u can have a major impact on the quality and characteristics of predictions. The main idea is that with $M \ll N$, we can reduce training time complexity. Namely, we approximate the latent function prior by assuming conditional independence between the training and test latent function values [13] (where q denotes an approximating distribution):

$$P(\mathbf{f}, \mathbf{f}_*|X, X_*) \approx q(\mathbf{f}, \mathbf{f}_*|X, X_*, X_u) = \int q(\mathbf{f}|\mathbf{u}, X, X_u)q(\mathbf{f}_*|\mathbf{u}, X_*, X_u)P(\mathbf{u}|X_u)d\mathbf{u}. \tag{2.10}$$

The prior on the inducing variables is exact: $P(\mathbf{u}|X_u) = \mathcal{N}(\mathbf{0}, K_{uu})$, but the conditional priors are simplified with further approximations. The exact forms for the two conditional priors are:

$$P(\mathbf{f}|\mathbf{u}, X, X_u) = \mathcal{N}(K_{fu}K_{uu}^{-1}\mathbf{u}, K_{ff} - Q_{ff}), \quad P(\mathbf{f}_*|\mathbf{u}, X_*, X_u) = \mathcal{N}(K_{*u}K_{uu}^{-1}\mathbf{u}, K_{**} - Q_{**}), \tag{2.11}$$

where we employ the convenient shorthand of $Q_{ab} = K_{au}K_{uu}^{-1}K_{ub}$ introduced in [13]. The FITC approximations $q(\mathbf{f}|\mathbf{u}, X, X_u)$ and $q(\mathbf{f}_*|\mathbf{u}, X_*, X_u)$ are as follows:

$$q(\mathbf{f}|\mathbf{u}, X, X_u) = \mathcal{N}(K_{fu}K_{uu}^{-1}\mathbf{u}, \text{diagm}(K_{ff} - Q_{ff})), \quad q(\mathbf{f}_*|\mathbf{u}, X_*, X_u) = P(\mathbf{f}_*|\mathbf{u}, X_*, X_u), \tag{2.12}$$

where the operator diagm(A) forms a diagonal matrix out of the diagonal elements in matrix A. This leads to the following prior over latent function values:

$$P(\mathbf{f}, \mathbf{f}_*|X, X_*) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} Q_{ff} + \text{diagm}(K_{ff} - Q_{ff}) & Q_{f*} \\ Q_{*f} & K_{**} \end{bmatrix}\right), \tag{2.13}$$

and the following predictive distribution:

$$P(\mathbf{f}_*|\mathbf{y}, X, X_u, X_*) = \mathcal{N}\left(Q_{*f}\left(Q_{ff} + \Lambda\right)^{-1}\mathbf{y}, K_{**} - Q_{*f}\left(Q_{ff} + \Lambda\right)^{-1}Q_{f*}\right), \tag{2.14}$$

where $\Lambda = \text{diagm}(K_{ff} - Q_{ff} + \sigma_n^2 I)$. Using the Sherman-Morrison-Woodbury formula for matrix inversion, the above expression can be rewritten so that the largest matrix inversion is $O(M^3)$. The resulting training time is $O(NM^2)$, and the prediction times are $O(M)$ and $O(M^2)$ for the mean and variance per test case (cf. Appendix A). The PITC approximation adds a slightly richer covariance structure to the conditional priors by replacing diagm$(\cdot)$ with blockdiagm$(\cdot)$, where blockdiagm(A) makes a block-diagonal matrix out of the elements in A. The sizes of the blocks can be as large as $M \times M$ to retain the computational time complexity for training. The hyperparameters (which now include locations of the inducing inputs X_u) are optimized with the marginal likelihood given by the model's new assumptions:

$$\log q(\mathbf{y}|X, X_u, \boldsymbol{\theta}) = -\frac{1}{2}\log\det\left(Q_{ff} + \Lambda\right) - \frac{1}{2}\mathbf{y}^T\left(Q_{ff} + \Lambda\right)^{-1}\mathbf{y} - \frac{N}{2}\log(2\pi), \tag{2.15}$$

with the appropriate definition of Λ for either FITC or PITC.

The Bayesian Committee Machine (BCM)

The BCM is a reformulation of PITC in which the blocks of data are physically distributed across different nodes, which is very much in the same spirit as our objectives. However, the BCM still uses a single set of globally optimized hyperparameters, which requires centralization before distributing the data across all nodes and also loses the flexibility of modeling with separate kernels in each node.

We begin deriving the BCM model by assuming that the inducing input locations and optimized hyperparameters are already given. To make notation consistent throughout our exposition, we will use the superscript $(\cdot)^G$ to denote "global" variables or terms, as opposed to $(\cdot)^i$ which will denote variables local to node i. The BCM operates by first predicting the values of the inducing variables and uses this result to perform predictions at other locations. This is simply another way of stating the approximation that the training and test latent values interact only through the inducing inputs. The distribution of the inducing variables is given by (with the conditioning

on $\hat{\boldsymbol{\theta}}$ omitted):

$$P(\mathbf{u}|\mathbf{y}, X_u, X) \propto P(\mathbf{u}|X_u)P(\mathbf{y}|X, X_u, \mathbf{u}). \tag{2.16}$$

This exact expression is then approximated using the PITC inducing conditional. Specifically, we assume that the blocks of blockdiagm$(K_{ff} - Q_{ff})$ correspond to a physically explicit partitioning of data among nodes. In other words, the latent function values of each node are conditionally independent of those in other nodes given $\mathbf{u}$. In this way, we can approximate the above expression by the following [25]:

$$\hat{P}(\mathbf{u}|X, X_u, \mathbf{y}) \propto P(\mathbf{u}|X_u) \prod_{i=1}^{m} P(\mathbf{y}^i|X^i, X_u, \mathbf{u}) \propto \frac{1}{P(\mathbf{u}|X_u)^{m-1}} \prod_{i=1}^{m} P(\mathbf{u}|\mathbf{y}^i, X^i, X_u), \tag{2.17}$$

where m indicates the number of nodes. The distribution $P(\mathbf{u}|\mathbf{y}^i, X^i, X_u)$ is simply the prediction of the local node i at the locations X_u. In the standard formulation of the BCM, each node is a standard GP, whereby this expression is given by Equation 2.6. However, this need not be the case, and each local node could itself be some other sort of Gaussian process, such as a sparse GP. This indicates the power of the BCM in generalizing PITC. Denote the posterior $P(\mathbf{u}|\mathbf{y}^i, X^i, X_u)$ as the distribution $\mathcal{N}(\mu^i(\mathbf{u}), \Sigma^i(\mathbf{u}))$. The entire expression above is then the product and quotient of Gaussian distributions, so it is easily obtained as:

$$\hat{P}(\mathbf{u}|X, X_u, \mathbf{y}) = \mathcal{N}(\mu^G(\mathbf{u}), \Sigma^G(\mathbf{u})), \tag{2.18a}$$

$$\Sigma^G(\mathbf{u})^{-1} = -(m-1)K_{uu}^{-1} + \sum_{i=1}^{m} \Sigma^i(\mathbf{u})^{-1}, \tag{2.18b}$$

$$\mu^G(\mathbf{u}) = \Sigma^G(\mathbf{u}) \sum_{i=1}^{m} \left(\Sigma^i(\mathbf{u})^{-1}\mu^i(\mathbf{u})\right). \tag{2.18c}$$

Computation of this global prediction requires the collection and addition of the local precision matrices as well as the precision-weighted means. In a network setting, this represents the only global step in the training process, and collection of these values will often be done by the master or leader node. Afterwards, prediction at test locations can be computed solely by the leader node without any need for communication to other nodes. Namely, the prediction at test locations is determined by the following expression:

$$\begin{aligned} \hat{P}(\mathbf{f}_*|\mathbf{y}, X, X_u, X_*) &= \int P(\mathbf{f}_*|\mathbf{u}, X_*, X_u)\hat{P}(\mathbf{u}|X, X_u, \mathbf{y})d\mathbf{u} \\ &= \mathcal{N}(\mu^G(\mathbf{f}_*), \Sigma^G(\mathbf{f}_*)), \end{aligned} \tag{2.19a}$$

$$\mu^G(\mathbf{f}_*) = K_{*u}K_{uu}^{-1}\mu^G(\mathbf{u}), \tag{2.19b}$$

$$\Sigma^G(\mathbf{f}_*) = K_{**} - Q_{**} + K_{*u}K_{uu}^{-1}\Sigma^G(\mathbf{u})K_{uu}^{-1}K_{*u} \tag{2.19c}$$

This expression is actually equivalent to the PITC predictive distribution, as we show in Appendix B. As noted above, the BCM is very close to satisfying our objective of decentralization and scalability. However, it loses some flexibility by assuming that all nodes have the same hyperparameters. Also, choosing the inducing variables via a centralized optimization procedure is not viable in our framework. In its original form, the BCM assumed that the inducing inputs were the test inputs, making it a transductive inference method [25]. We will introduce a more general approach below that also accounts for heterogeneous hyperparameters.

2.2 The Bayesian Committee Machine of Experts (BCME) Model

We now generalize the BCM to include localized or heterogeneous hyperparameters. We assume that the X_u are hyperparameters, and to assert scalability, we assign responsibility of determining the locations of a subset of these inducing inputs to each node.

Our approach is modeled largely after the BCM, and we begin by attempting to marginalize over the hyperparameters (including the inducing inputs). We will denote the aggregate vector of hyperparameters for *all* nodes as $\boldsymbol{\theta}$, and the local hyperparameters as $\boldsymbol{\theta}^i$. Similarly, the inducing inputs assigned to node i are denoted as X_u^i. Attempting to marginalize out the hyperparameters and inducing inputs yields:

$$\begin{aligned} P(\mathbf{u}|\mathbf{y}, X) &= \int P(\mathbf{u}|\mathbf{y}, X, X_u, \boldsymbol{\theta}) P(X_u, \boldsymbol{\theta}|\mathbf{y}, X) d\boldsymbol{\theta} dX_u \\ &= \frac{1}{P(\mathbf{y}|X)} \int P(\mathbf{u}|X, X_u, \boldsymbol{\theta}) P(\mathbf{y}|\mathbf{u}, X, X_u, \boldsymbol{\theta}) P(X_u, \boldsymbol{\theta}|X) d\boldsymbol{\theta} dX_u \end{aligned} \tag{2.20}$$

We introduce the approximating distribution, which now includes the factorization of the posterior $P(\mathbf{y}|\mathbf{u})$ as well as the hyperprior $P(X_u, \boldsymbol{\theta})$:

$$\begin{aligned} \hat{P}(\mathbf{u}|\mathbf{y}, X) &\propto \int P(\mathbf{u}|X, X_u, \boldsymbol{\theta}) \left(\prod_{i=1}^m P(\mathbf{y}^i|\mathbf{u}, X^i, X_u, \boldsymbol{\theta}^i) P(X_u^i, \boldsymbol{\theta}^i|X^i) \right) d\boldsymbol{\theta} dX_u \\ &= \int P(\mathbf{u}|X, X_u, \boldsymbol{\theta}) \left(\prod_{i=1}^m \frac{P(\mathbf{u}|\mathbf{y}^i, X^i, X_u, \boldsymbol{\theta}^i) P(\mathbf{y}^i|X^i, X_u, \boldsymbol{\theta}^i)}{P(\mathbf{u}|X^i, X_u, \boldsymbol{\theta}^i)} P(X_u^i, \boldsymbol{\theta}^i|X^i) \right) d\boldsymbol{\theta} dX_u \end{aligned} \tag{2.21}$$

Furthermore, we stipulate that the local marginal likelihoods can only depend on the local inducing inputs so that $P(\mathbf{y}^i|X^i, X_u, \boldsymbol{\theta}^i) = P(\mathbf{y}^i|X^i, X_u^i, \boldsymbol{\theta}^i)$, whereby the local marginal likelihoods can be computed in a completely decentralized fashion. Now we must determine how to optimize the inducing inputs and hyperparameters. If each local node is a standard GP, then the local marginal likelihoods will be independent of the inducing inputs. In this case, we need to include regularization term that augments the standard marginal likelihood to find $\boldsymbol{\theta}^i$ as well as X_u^i. We will discuss this regularization term shortly. If we assume that the prior is again flat or vague and that $P(\mathbf{y}^i|X^i, X_u^i, \boldsymbol{\theta}^i)$ is highly peaked at $\hat{\boldsymbol{\theta}}^i, \hat{X}_u^i$, then we have:

$$\hat{P}(\mathbf{u}|\mathbf{y}, X) \approx \hat{P}(\mathbf{u}|\mathbf{y}, X, \hat{X}_u, \hat{\boldsymbol{\theta}}) \propto P(\mathbf{u}|X, \hat{X}_u, \hat{\boldsymbol{\theta}}) \prod_{i=1}^m \frac{P(\mathbf{u}|\mathbf{y}^i, X^i, \hat{X}_u, \hat{\boldsymbol{\theta}}^i)}{P(\mathbf{u}|X^i, \hat{X}_u, \hat{\boldsymbol{\theta}}^i)}. \tag{2.22}$$

Note that we have a very similar expression to the BCM except that we multiply on the left by a *global* prior on inducing variables that takes into account all input data points and all hyperparameters, whereas we divide the local posteriors by local priors. This formulation is ensured to be a valid distribution since the local posterior divided by the local prior yields a distribution with a positive definite covariance. The prediction on inducing variables is:

$$\hat{P}(\mathbf{u}|\mathbf{y}, X, \hat{X}_u, \hat{\boldsymbol{\theta}}) = \mathcal{N}(\mu^G(\mathbf{u}), \Sigma^G(\mathbf{u})), \tag{2.23a}$$

$$\Sigma^G(\mathbf{u})^{-1} = (K_{uu}^G)^{-1} + \sum_{i=1}^m \left(\Sigma^i(\mathbf{u})^{-1} - (K_{uu}^i)^{-1} \right), \tag{2.23b}$$

$$\mu^G(\mathbf{u}) = \Sigma^G(\mathbf{u}) \sum_{i=1}^{m} \Sigma^i(\mathbf{u})^{-1} \mu^i(\mathbf{u}). \tag{2.23c}$$

Similarly, the prediction on test inputs is:

$$\begin{aligned} \hat{P}(\mathbf{f}_*|\mathbf{y}, X, \hat{X}_u, X_*, \hat{\boldsymbol{\theta}}) &= \int P(\mathbf{f}_*|\mathbf{u}, X_*, \hat{X}_u, \hat{\boldsymbol{\theta}}) \hat{P}(\mathbf{u}|X, X_u, \mathbf{y}, \hat{\boldsymbol{\theta}}) d\mathbf{u} \\ &= \mathcal{N}(\mu^G(\mathbf{f}_*), \Sigma^G(\mathbf{f}_*)), \end{aligned} \tag{2.24a}$$

$$\mu^G(\mathbf{f}_*) = K^G_{*u}(K^G_{uu})^{-1}\mu^G(\mathbf{u}), \tag{2.24b}$$

$$\Sigma^G(\mathbf{f}_*) = K^G_{**} - Q^G_{**} + K^G_{*u}(K^G_{uu})^{-1}\Sigma^G(\mathbf{u})(K^G_{uu})^{-1}K^G_{u*} \tag{2.24c}$$

where we have set the distribution $P(\mathbf{f}_*|\mathbf{u}, X_*, \hat{X}_u, \hat{\boldsymbol{\theta}})$ to depend on the global kernel just as with the global prior.

We still have to determine two crucial aspects of this model. First, we need to determine how to augment the marginal likelihood so that each node can optimize both hyperparameters and inducing input locations. This is particularly important when each local node is a standard GP, but it can also be relevant to situations when local nodes are sparse GPs that have a dependence on inducing inputs. Indeed, we now have the ability to have both local and global inducing inputs, i.e. inducing inputs that nodes share with others as well as inducing inputs that they only use for local predictions.

The second aspect concerns the global kernel. Note that, in the limiting case where all of the local hyperparameters are the same, we would like the global kernel to also assume the common homogeneous kernel, reducing the system to a standard BCM. In the case of highly heterogeneous kernels, we need the global kernel to accurately map the prior uncertainties of local nodes on test inputs. One consideration is that the global kernel uses hyperparameters learnt on a subset of data; another possibility is that the kernel function is a weighted average of local kernel functions, with weights taking into account which nodes are experts in certain regions of input space.

2.2.1 Loss Function for Inducing Inputs

Our approach to determining a loss function for the global inducing inputs differs from the approach of most sparse GP methods. Most methods assume that the inducing variable approximation is true (i.e. that latent function values are fully or partially independent given the inducing variables), and then they perform optimization of the associated approximate marginal likelihood (Equation 2.15). Our approach is to choose the inducing input locations to make the inducing variable approximation as accurate as possible. Of course, we will not allow any global communication during the optimization process, so this will be done in a heuristic fashion.

Intuitively, the inducing variable approximation is valid when each node's local inducing inputs/outputs act like a compressed representation of the node's actual data. In other words, we need the local inducing inputs/outputs of each node to describe the corresponding local data as completely as possible, such that given this compressed set, the other nodes learn as much as possible about the local data. When the compressed set completely describes the local data, the inducing variable "approximation" is actually an exact expression.

As a first thought for optimizing the local inducing input locations, we may consider trying to maximize the mutual information between $\mathbf{u}^i$ and $\mathbf{y}^i$, given by the following (with implicit

conditioning on X^i and X^i_u):

$$MI(\mathbf{u}^i, \mathbf{y}^i) = \int P(\mathbf{u}^i, \mathbf{y}^i) \log \left(\frac{P(\mathbf{u}^i, \mathbf{y}^i)}{P(\mathbf{u}^i)P(\mathbf{y}^i)} \right) d\mathbf{u}^i d\mathbf{y}^i$$
$$= \frac{1}{2} \log \det(K^i_{f^i f^i} + (\sigma^i_n)^2 I) - \frac{1}{2} \log \det(K^i_{f^i f^i} - K^i_{f^i u^i}(K^i_{u^i u^i})^{-1} K^i_{u^i f^i} + (\sigma^i_n)^2 I). \tag{2.25}$$

This loss function does not depend on actual target values but rather only on the input locations (and hyperparameters).[2] Instead of calculating this value, however, we opt for an even simpler approximation of this expression that is inspired by a variational approach to sparse GP regression [23]. We choose to minimize the expected squared error of predicting $\mathbf{f}^i$ using $\mathbf{u}^i$, normalized by the noise level of predicting $\mathbf{y}^i$ using $\mathbf{f}^i$:

$$\frac{1}{2(\sigma^i_n)^2} \int P(\mathbf{u}^i, \mathbf{f}^i) \| K^i_{f^i f^i} (K^i_{u^i u^i})^{-1} \mathbf{u}^i - \mathbf{f}^i \|^2 d\mathbf{u}^i d\mathbf{f}^i = \frac{1}{2(\sigma^i_n)^2} \operatorname{trace}(K^i_{f^i f^i} - K^i_{f^i u^i}(K^i_{u^i u^i})^{-1} K^i_{u^i f^i}). \tag{2.26}$$

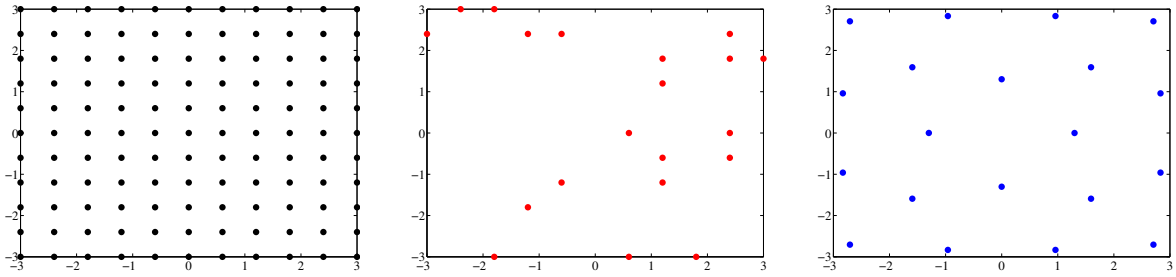
We can verify intuitive notions of this loss function's behavior with simple examples. In Figure 2.2 we show the input locations of a toy two-dimensional dataset consisting of 121 points. Note that the input data has a uniform distribution. We show the initial locations of inducing inputs (chosen to be a random subset of the data), as well as the optimized locations. The final inducing input locations reach a distribution that attempts to cover the input data distribution, which is precisely what we require.

In Figure 2.3, we verify the notion that this optimization heuristic attempts to make the inducing conditional approximation more accurate. We evenly split Silverman's motorcycle dataset [17] among 4 experts (Figure 2.3(a)). Each expert first optimizes its hyperparameters and then uses these values to optimize X^i_u. In our pre-established notational convention, $K^i_{ff} - Q^i_{ff}$ denotes the conditional covariance matrix for all data using $\boldsymbol{\theta}^i$.[3] We define the operator nonblockdiagm$(K^i_{ff} - Q^i_{ff})$ as the conditional covariance matrix with zeros at each block corresponding to a local node's data. In other words, nonblockdiagm$(K^i_{ff} - Q^i_{ff})$ looks only at the (conditional) covariances between data points on different nodes. The inducing conditional approximation is accurate when $\| \text{nonblockdiagm}(K^i_{ff} - Q^i_{ff}) \|_F$ is small. In Figure 2.3(b), we see that each node's optimization procedure makes the off-block regions smaller, although not monotonically. It is important to recognize that the optimization scheme only has access to each node's local data points, so directly minimizing the norm of the off-block regions is impossible.

At this point, we also need to consider whether to jointly or separately optimize the global inducing inputs and the local hyperparameters $\boldsymbol{\theta}^i$. As noted in [23], joint optimization has a regularizing effect on $\boldsymbol{\theta}^i$. If each node is a sparse GP such that the local marginal likelihood depends on the inducing inputs, this issue is not of great concern. However, it is of major concern when each node is a full GP model. We would expect the regularizing effect to be negative when the nodes have random subsets of data, as each node will reach an over-smoothed solution. The case of clustered data is not as clear. We have found that separate optimization is both faster and leads to more accurate solutions in the case where each node is a full GP (cf. Appendix C).

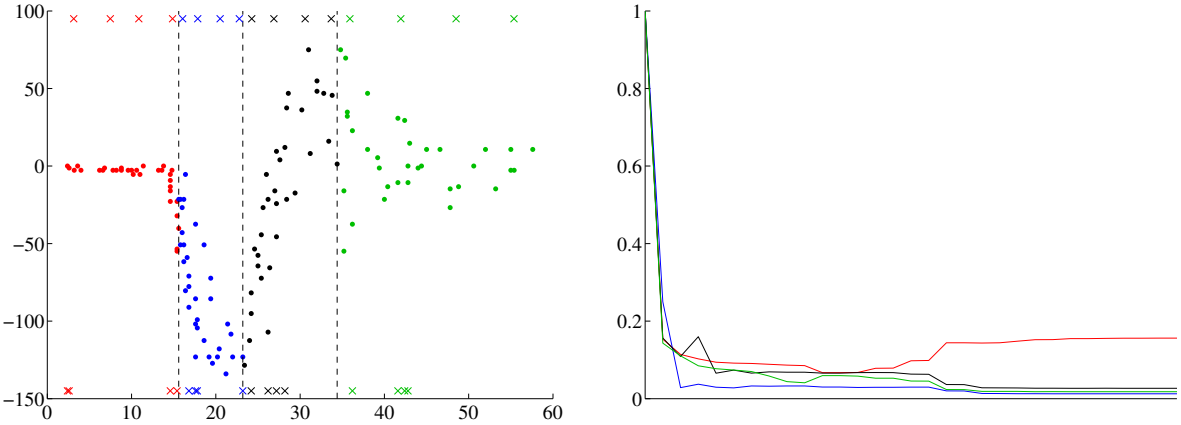
[2] Note that $K^i_{f^i u^i}(K^i_{u^i u^i})^{-1} K^i_{u^i f^i} \neq Q^i_{f^i f^i} := K^i_{f^i u}(K^i_{uu})^{-1} K^i_{u f^i}$. The expression for $Q^i_{f^i f^i}$ depends on all X_u, not just X^i_u.

[3] This covariance matrix can only be calculated in a centralized fashion, since it requires knowledge of the locations of all data points from all experts.



(a) Input data (b) Initial X_u (c) Optimized X_u

Figure 2.2. Optimization of X_u over a toy dataset. 20 inducing inputs are optimized in a dataset of 121 points. Initial X_u locations are chosen as a random subset of the input data. The optimized locations attempt to match the distribution of input data, resulting in a symmetry pattern.



(a) Acceleration (g's) vs. time (milliseconds) for motorcycle impact.

(b) $\|\,\text{nonblockdiagm}(K^i_{ff} - Q^i_{ff})\|_F$ throughout optimization procedure

Figure 2.3. Optimization of X_u over the motorcycle dataset (skull acceleration (g) vs. time (ms) during a simulated motorcycle accident). In (a), we show the partitioning of data amongst the 4 nodes, depicted by the 4 colors. Initial X_u locations (4 per node) are shown as crosses at the bottom, and optimized locations are shown at the top. In (b), we show the value of $\|\,\text{nonblockdiagm}(K^i_{ff} - Q^i_{ff})\|_F$ as each node simultaneously optimizes X^i_u. The values are normalized by $\|\,\text{nonblockdiagm}(K^i_{ff} - Q^i_{ff})\|_F$ at the initialization of each optimization procedure such that all values begin at 1. The color scheme corresponds to the same nodes as in (a).

2.2.2 Gating Function for the Global Kernel

The global kernel determines the global prior over input locations. In the limit where all of the nodes cover random subsets of data (and thus have roughly the same hyperparameters), the global kernel should also assume the same form. However, in the cases where the nodes truly are experts in different regions of input space (and thus have different sets of hyperparameters), we would like the global kernel to weight experts' priors based on their proximities to input locations. This specification naturally leads to a gating function mechanism, but we cannot simply weight covariance matrices by a single scalar. Rather, we must multiply all points by a deterministic function. Furthermore, we require that each node must be able to compute its local contribution

to the gating function independently of all others. As such, we define the global kernel as:

$$k^G(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{m} \frac{g^i(\mathbf{a}) k^i(\mathbf{a}, \mathbf{b}) g^i(\mathbf{b})}{\sqrt{\sum_{i=1}^{m} g^i(\mathbf{a})^2} \sqrt{\sum_{i=1}^{m} g^i(\mathbf{b})^2}}, \tag{2.27}$$

where $g^i(\cdot)$ is the local gating function for each node. Since the leader node already collects precision matrices and precision-weighted means from the other nodes, it can also easily collect and sum the gating functions to form the normalization constant in the denominator. Now we need to consider the actual form for $g^i(\cdot)$. We adopt the convention of [10] and let $g^i(\cdot)$ be given by the following:

$$g^i(\mathbf{a}) = \frac{k^i(\mathbf{a}, \bar{\mathbf{x}}^i)}{k^i(\mathbf{a}, \mathbf{a})}, \tag{2.28}$$

where $\bar{\mathbf{x}}^i$ is the centroid of node i's data. We normalize by the autocorrelation intensity (i.e. $(\sigma_f^i)^2$) so that only the characteristic length scales determine the strength of an expert on an input. Otherwise, nodes that are more uncertain (i.e. with stronger signal magnitudes) would be weighted strongly by the gating mechanism, as shown in Figure 2.4. Unlike the approach of [9], which uses a gating mechanism similar to Mahalanobis distance with respect to the input data distribution, our approach and that of [10] use the learnt length scales to determine proximity of experts to data locations. This approach takes further advantage of the heterogeneous hyperparameters in deciding how to weight experts' predictions, as opposed to using distances that depend solely on input data distributions alone. It also yielded better accuracies on test datasets.

We can set global noise levels using the same gating function so that we get the predictive distribution on noisy targets:

$$\hat{P}(\mathbf{y}_*|\mathbf{y}, X, \hat{X}_u, X_*, \hat{\boldsymbol{\theta}}) = \mathcal{N}(\mu^G(\mathbf{y}_*), \Sigma^G(\mathbf{y}_*)), \tag{2.29a}$$

$$\mu^G(\mathbf{y}_*) = \mu^G(\mathbf{f}_*), \tag{2.29b}$$

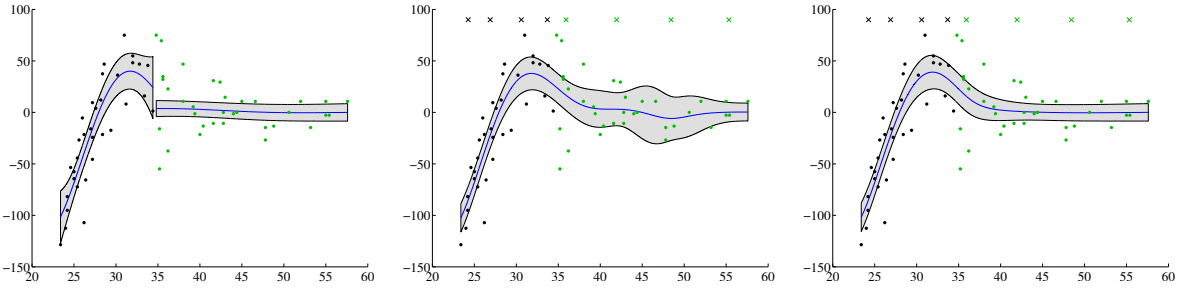
$$\Sigma^G(\mathbf{y}_*) = \Sigma^G(\mathbf{f}_*) + \text{diagm}(\boldsymbol{\sigma}_n^G(\mathbf{y}_*)(\boldsymbol{\sigma}_n^G(\mathbf{y}_*))^T), \tag{2.29c}$$

where $\boldsymbol{\sigma}_n^G$ is the vector of (now heterogeneous) noise levels at all target locations:

$$(\sigma_n^G(y_*))^2 = \sum_{i=1}^{m} \frac{g^i(\mathbf{x}_*)^2 (\sigma_n^i)^2}{\sum_{i=1}^{m} g^i(\mathbf{x}_*)^2}. \tag{2.29d}$$

2.2.3 Implementation in a Network Setting and Time Complexities

The BCME model requires no global step in hyperparameter training, as each node locally optimizes its hyperparameters $\boldsymbol{\theta}^i$ and inducing inputs X_u^i. This is the most time consuming step of the entire GP model, since it requires a gradient-based optimization procedure that inverts matrices in every iteration. It is orders of magnitude more time consuming than training and prediction (i.e. at least minutes as opposed to seconds in our experiments). Suppose that the i-th node has N^i data points, such that $\sum_{i=1}^{m} N^i = N^G$ is the total number of data points. We allocate the number of local inducing inputs such that the i-th node has MN^i/N^G local inducing inputs, which reduces to M/m in the case where all nodes have the same number of data points. If each node is a standard GP, hyperparameter training time for the i-th node is $O(\max(MN^i/N^G, N^i)^3) = O((N^i)^3)$ since $M/N^G \leq 1$. This yields an overall hyperparameter training time of $O(\max_i (N^i)^3)$.



(a) Local expert predictions (b) BCME, $g^i(\mathbf{a}) = k^i(\mathbf{a}, \bar{\mathbf{x}}^i)$ (c) BCME, $g^i(\mathbf{a}) = \frac{k^i(\mathbf{a}, \bar{\mathbf{x}}^i)}{k^i(\mathbf{a}, \mathbf{a})}$

Figure 2.4. Gating function behavior for a subset of the motorcycle dataset from Figure 2.3(a). (a) shows the (noise-free) predictions of two experts over their own respective regions of the input domain. (b) shows the (noise-free) predictions of the BCME when the gating function includes the signal magnitude in $g^i(\cdot)$. The left expert's predictions are considered too strongly over points $> 40ms$, as indicated by the large uncertainty and meandering behavior. (c) shows the (noise-free) predictions of the BCME with $g^i(\cdot)$ normalized by the signal magnitude. The left expert's predictions no longer overpower the right expert's predictions for inputs $> 40ms$. Inducing input locations are shown as crosses at the top of (b) and (c).

Training is a global procedure, since we collect and then disseminate all X_u locations to every node, and then we collect values from local nodes to compute the global prediction $\hat{P}(\mathbf{u}|\mathbf{y}, X, \hat{X}_u, \hat{\boldsymbol{\theta}})$. However, this is just a single step as opposed to an iterative procedure, so training is orders of magnitude faster than hyperparameter training. The most intuitive network configuration for training is a star, with a leader node connected to all other nodes, but this structure is not necessary. In more general networks, we still identify a leader node (cf. Section 3.4.4), but other nodes need not be connected to it directly. They can simply pass values along a shortest path tree so that communication with the global node occurs via intermediate nodes (and sums are computed via intermediate sums). Each node computes four local values to communicate to the global node: $\Sigma^i(\mathbf{u})^{-1} - (K^i_{uu})^{-1}$, $\Sigma^i(\mathbf{u})^{-1}\mu^i(\mathbf{u})$, the vector of gating function values at X_u, and the local component of gated prior (i.e. the numerator of Equation 2.27). The overall time complexity for these operations is $O(\max(M, N^i)^3)$. Communicating these values from all nodes to the leader has a worst case time complexity of $O(mM^2) \leq O(M^3)$, which occurs in a star or chain network configuration. In tree-structured networks, this cost can be as low as $O(M^2 \log m)$. Finally, the global node computes the denominator of Equation 2.27 to form K^G_{uu}, whereby it can compute the relevant precomputations for $\hat{P}(\mathbf{f}_*|\mathbf{y}, X, \hat{X}_u, \hat{\boldsymbol{\theta}})$. The computational cost within the global node for these operations is $O(M^3)$. Thus, assuming that each node performs its local computations in parallel, we have an overall training time complexity of $O(\max(M, \max_i(N^i))^3)$.

Prediction time for the mean of $\hat{P}(\mathbf{y}_*|\mathbf{y}, X, \hat{X}_u, \hat{\boldsymbol{\theta}})$ is dominated by computation of K^G_{*u}. This can be done in two ways. In the centralized method, we assume that during training time the global node collected all $\boldsymbol{\theta}^i$ from other nodes so that it can compute the local components K^i_{*u}; this is $O(mM) \leq O(M^2)$ per test case. In the distributed method, the global node distributes X_* to all other nodes and collects K^i_{*u}, which has a worst case time complexity of $O(mM)$ for star and chain configurations, but can be as low as $O(M \log m)$ for a tree-structured network. Once K^G_{*u} is computed, computation of the latent function uncertainty can be done entirely by the global node. As with computation of K^G_{*u}, computation of the global noise levels $\sigma^G_n(y_*)$ can be done in either a centralized fashion if the global node knows all $\boldsymbol{\theta}^i$ or in a decentralized fashion. Either way, the time complexity is dominated by matrix multiplications in Equation 2.24c, which are $O(M^2)$ per test case. Thus, prediction has an overall time complexity of $O(M^2)$ per test case.

Table 2.1. Dataset Properties

Dataset	Input dimensions (d)	Training Size (N^G)	Test Size
Abalone	8	3133	1044
KIN40K	8	10000	30000
SARCOS	21	44484	4449

Table 2.2. Experiment Parameters

Dataset	Number of Nodes				M/N^G (%)				
Abalone	2	4	8	16	1	5	10	25	50
KIN40K	5	10	20	40	0.5	1	5	10	20
SARCOS	25	50	100	200	0.25	0.5	1	2	4

2.3 Experimental Evaluations

We now focus on experimentally comparing the BCME model with other methods. Importantly, all of the other methods use a centralized optimization approach to determining hyperparameters and inducing input locations. We find that our method remains competitive for given computational and/or time budgets. For all methods, we use a scaled nonlinear conjugate gradient algorithm for training hyperparameters and inducing inputs, and we cap the total number of optimization iterations at 1000.

We use the Abalone, KIN40K, and SARCOS datasets, which are summarized in Table 2.1; these datasets are relatively popular in evaluating sparse GP methods. The Abalone dataset requires the prediction of age of abalone given 8 characteristics, and it is known to have low complexity and high noise. The KIN40K dataset requires the prediction of the distance of a robot arm from a target given various joint positions and rotations. Finally, the SARCOS dataset is an inverse dynamics problem requiring the prediction of the torque at a robot joint given the positions, velocities, and accelerations of 7 points on the arm. The latter two datasets are highly nonlinear.

The experimental parameters are shown in Table 2.2. We standardize the datasets to have zero mean and unit variance for each input dimension of the training set and zero mean for the output of the training set. For each dataset, we run experiments with 4 values for the number of local nodes as well as 5 values for the number of inducing inputs. As the datasets grow in size, we use more nodes and fewer inducing inputs. The largest cases we consider are when local nodes have ≈ 2000 to 3000 data points and $M \approx 2000$. To showcase the full capabilities of the BCME, we cluster the data using k-means clustering, whereby the number of local inducing points for node i to optimize is MN^i/N^G. In addition, we separately optimize $\boldsymbol{\theta}$ and X_u. The reasoning behind these choices are fully explained in Appendix C, where we evaluate the effects of the X_u optimization procedure and clustering of data on error measures.

We compare our model's accuracy with two centralized sparse models: FITC and the variational sparse GP presented in [23] (abbreviated as Var). We evaluate the centralized methods at the same values for M/N^G used for the BCME experiments (Table 2.2).[4] Additionally, we use a subset of data (SoD) approach with 2000 data points and a linear regression (LR) model as baseline comparisons.

[4]For SARCOS, we do not evaluate FITC and Var at $M/N^G = 4\%$ due to time limitations.

2.3.1 Error Measures

Throughout our exposition, we will employ two accuracy measures: the standardized mean square error ($SMSE$), and standardized negative log probability ($SNLP$) of test data. Let a test latent function value be denoted by $\mathbf{f}_s$, a test target as y_s, the vector of training targets as $\mathbf{y}_T$ and the vector of test targets as $\mathbf{y}_S$. Then the SMSE is given by:

$$SMSE = \frac{E\left[\left(y_s - \mu^G(\mathbf{f}_s)\right)^2\right]}{\mathrm{Var}(\mathbf{y}_S)}, \tag{2.30}$$

where $\mathrm{Var}(\mathbf{y}_S)$ is the variance of the test targets, and $E[\cdot]$ is an expected value (in this case the average over test cases). Simply predicting the mean of the training targets yields an SMSE of roughly 1, and more accurate methods have values closer to 0. Now denote $V^G(y_s)$ as the sum of the uncertainty in the latent function value and the noise. For the BCME, this is:

$$V^G(y_s) = \mathbf{e}_s^T \Sigma^G(\mathbf{y}_S)\mathbf{e}_s, \tag{2.31}$$

where $\mathbf{e}_s$ is a unit vector with 1 at entry s. Then the SNLP is given by:

$$SNLP = NLP - NLP_0, \tag{2.32a}$$

$$NLP = \frac{1}{2}E\left[\log(2\pi V^G(y_s)) + \frac{\left(y_s - \mu^G(\mathbf{f}_s)\right)^2}{V^G(y_s)}\right] \tag{2.32b}$$

$$NLP_0 = \frac{1}{2}\log(2\pi\,\mathrm{Var}(\mathbf{y}_T)) + \frac{1}{2}E\left[\frac{(y_s - E[\mathbf{y}_T])^2}{\mathrm{Var}(\mathbf{y}_T)}\right]. \tag{2.32c}$$

Simply predicting the mean of the training targets with an uncertainty given by their variance yields an SNLP of 0. More accurate methods yield $SNLP < 0$.

2.3.2 Error Measures vs. M

Figure 2.5 compares the error metrics for all models with M (SoD and LR are shown as horizontal lines because these methods are independent of M).[5] In general, all methods that depend on inducing points improve with M, as expected. However, the relative accuracy of the different methods varies by dataset, as we now explain.

Abalone is evidently almost linear, as the GP methods are not able to perform much better than the LR in terms of $SMSE$. FITC significantly outperforms all other methods in terms of $SNLP$, but this occurs surprisingly at the smallest M and has been previously noted to be a result of the extra flexibility inherent in FITC models [20]. Beyond the extreme limit of small M, the BCME is competitive, and it even slightly outperforms the other methods at large M.

KIN40K yields much more interesting results due to the high degree of nonlinearity present in the data: LR performs just as poorly as always predicting 0 for the outputs. There are two important observations regarding the trends for the BCME models. First, variations in accuracy with m appear to be negligible. Intuitively, large m with clustered data implies that each node has a small subset of data points representing a small region of the input domain. This characteristic can make nodes prone to overfitting hyperparameters. Recall that predictions

[5] For LR, we calculate $SNLP$ values using confidence intervals of the regression coefficients.

on test inputs occur via predictions on X_u. Thus, the effects of overfitting hyperparameters are most evident for small M, since the distances to interpolate between test and inducing inputs are high. Increasing M decreases the interpolation distances, which can assuage (if not completely mitigate) the overfitting problem. For KIN40K, the overfitting problem is only slightly evident in the $SMSE$ even at the smallest value of M tested. This likely implies that KIN40K is fairly homoscedastic and homogeneous in the length scales of its dynamics. Second, we find that the BCME models begin with much worse accuracy (i.e. higher error values) than FITC, Var, and SoD, but they have much faster convergence rates with M. In other words, the BCME models are able to "catch up" to the centralized sparse models: at $M = 2000$ (i.e. $M/N^G = 20\%$), the $SMSE$ values are the same as that of FITC, and the $SNLP$ values have almost reached those of Var.

The SARCOS dataset likely has a higher signal-to-noise ratio than KIN40K since the baseline $SMSE$ values are significantly smaller. It is still clearly nonlinear, as LR performs much worse than SoD. For the BCME models, the stratification in accuracy with m (at any given M) is more prominent than with KIN40K, indicating that overfitting is more of a concern with SARCOS. In addition, we again see faster convergence rates for the BCME models with M compared to FITC and Var, but we have not chosen M large enough for the BCME models to fully "catch up," as observed in KIN40K. Both of these observations indicate that we should attempt larger M for the BCME models. Based on results for Abalone and KIN40K, we see that 10% to 20% roughly appears to be the value for M/N^G at which the BCME models become competitive with the centralized sparse models. We will revisit this observation in Section 2.3.3 below.

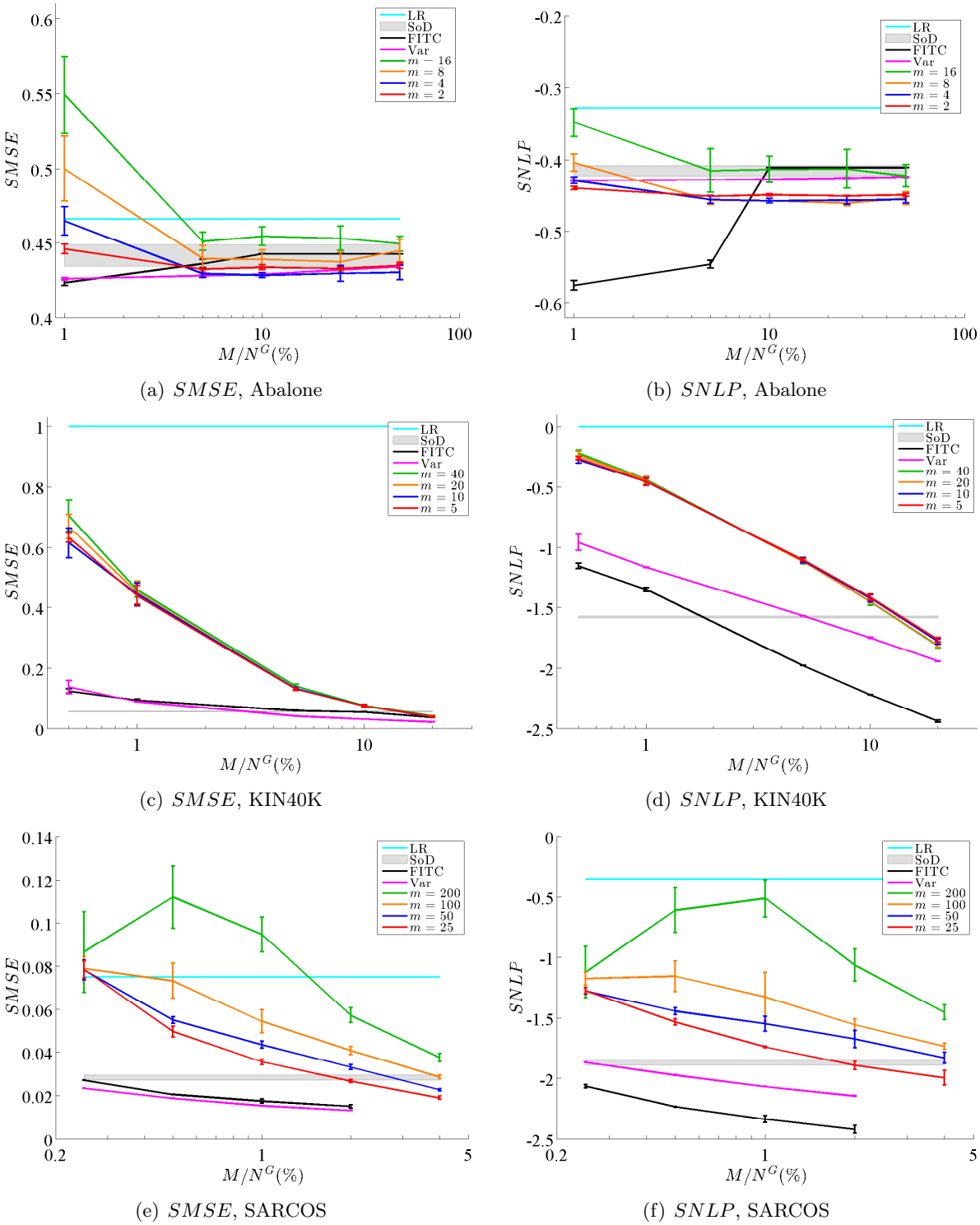
First, we need to point out a crucial flaw in the results illustrated in Figure 2.5. For centralized sparse models, M gives an indication of the computational complexity involved with hyperparameter training. This is not the case with the BCME, as changes in m for a given M can drastically alter computational costs. If our aim is to compare accuracy with respect to computational and time budgets, plotting error measures with respect to M/N^G no longer makes sense.

2.3.3 Error Measures vs. Hyperparameter Training Time

We now examine errors with respect to both theoretical and empirical hyperparameter training time. For the BCME models, we plot only the best accuracy for each m (i.e. the cases with the largest M). We plot only GP methods; LR requires about 3 seconds or less to train for each of the datasets tested, whereas all other methods require time at least on the order of minutes.

Figure 2.6 shows errors with theoretical hyperparameter training time. Although the overall complexities are $O(N^G M^2)$ for FITC/VAR and $O(\max_i(N^i)^3)$ for the BCME, many other lower-order operations can change the overall computation time. Thus, for the theoretical time we compute the time complexity of one iteration of the conjugate gradient algorithm, including all operations that are third-order (or higher) in the variables N^G (for centralized methods), N^i (for the BCME), M, and/or d (the dimensionality of the data). This includes, for example, operations such as computing the covariance matrix between inducing inputs, which is $O(dM^2)$.

This measure for computational cost is accurate only if the following two assumptions hold: there are no other sources of significant computational overhead and latency, and all methods require the same number of iterations to converge. In practice, the first assumption is rarely true, and the second is often violated as well. For this reason, we also plot the error measures with empirical hyperparameter training times in Figure 2.7. The times were calculated by running experiments on a distributed computing environment that is shared with other users; therefore,



(a) $SMSE$, Abalone (b) $SNLP$, Abalone

(c) $SMSE$, KIN40K (d) $SNLP$, KIN40K

(e) $SMSE$, SARCOS (f) $SNLP$, SARCOS

Figure 2.5. Error measures vs. M/N^G for all methods (SoD and LR are independent of M). Error bars extend $\pm$ one standard deviation from the mean value over five trials. The BCME methods are colored according to network size (i.e. m).

there were greatly varying loads on memory and processor resources which added further latency and overhead to hyperparameter training times.

When comparing Figures 2.6 and 2.7, we see that the trends for errors are qualitatively similar, but the scale of variation is much smaller in the empirical times. For example, Figure 2.6(c) and (d) predict that the BCME with $m = 40$ is roughly 5 orders of magnitude faster than FITC with the largest value of M for KIN40K. In reality, Figure 2.7(c) and (d) show that it was only about 3 orders of magnitude faster. Otherwise, if we ignore the scales of the time axes, the different methods fall roughly in the same locations relative to each other. The only major anomaly is the training time of the SoD method for KIN40K, which converged in very few iterations.

Overall, we see that the BCME models are able to achieve competitive error values at either the same (for small m) or mere fractions (for large m) of the computational cost associated with centralized methods. As noted above, the BCME has not yet "caught up" with the other methods for the SARCOS dataset with the M values we have tested. This clearly begs the question as to whether we can push the BCME's accuracy simply by increasing M, which we consider next.

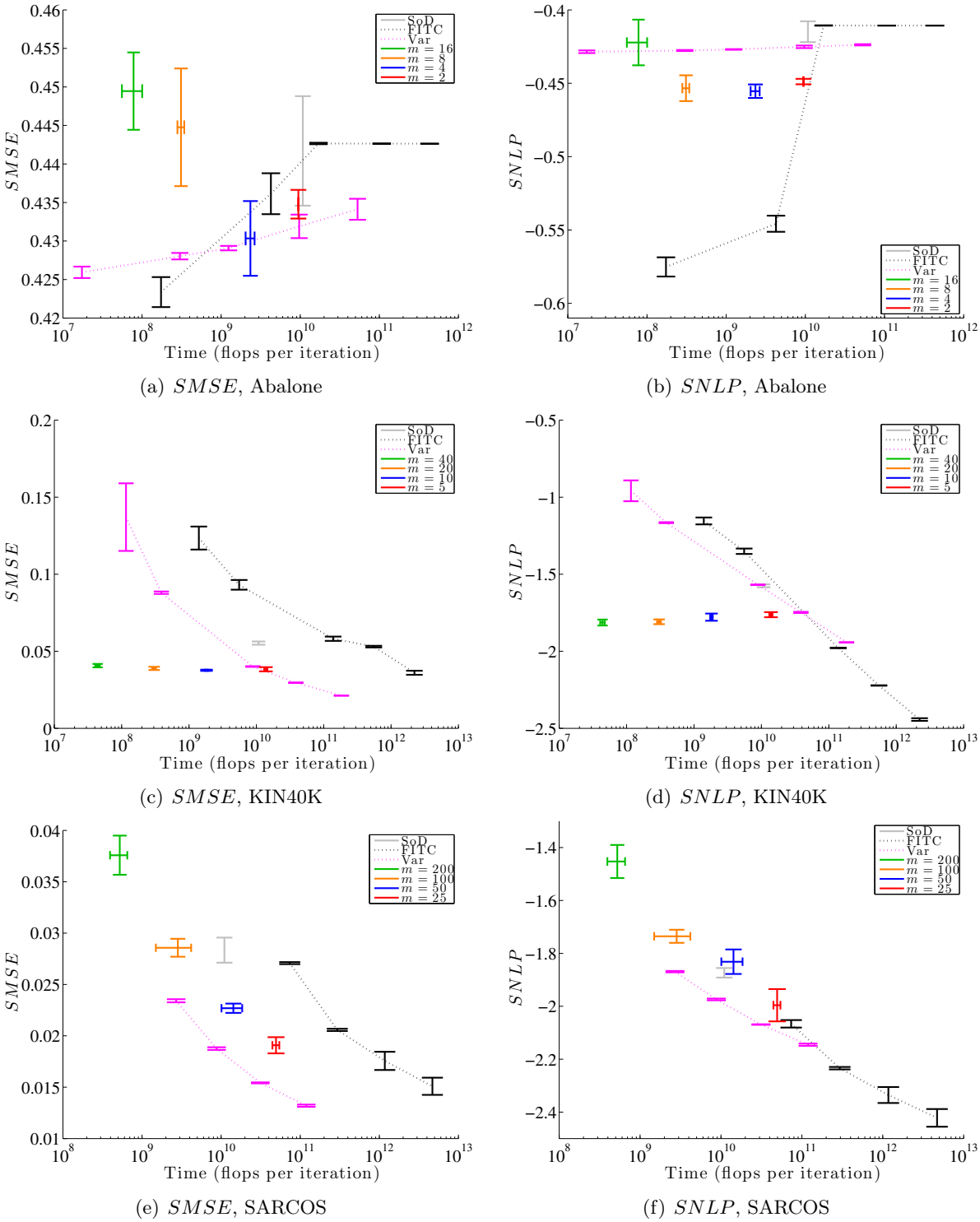
2.3.4 Pushing the Limits of the BCME

We test the cases of $m = 100$ and $m = 200$ with $M/N^G = 8\%$, 10%, and 12%. Figures 2.8(a) and 2.8(b) recreate Figures 2.7(e) and 2.7(f) with these appended cases and with the BCME models of $m = 50$ and $m = 25$ at $M/N^G = 4\%$ removed. The BCME models are able to "catch up" in terms of reaching competitive $SMSE$ and $SNLP$ values while retaining computational advantages. Indeed, we are able to match Var's accuracy at $M/N^G = 1\%$ roughly 100 times faster using $m = 100$. Equivalently, given a tight budget (i.e. less than 100 minutes), the BCME is the method of choice, as it is the only method capable of providing reasonable results within this budget. We can also conclude that $M/N^G = 10\%$ to 20% resulted in competitive results for the BCME with our data; it is worthwhile to see whether this scaling holds for other datasets.

Importantly, we cannot keep increasing M and m and expect the accuracy gains to continue indefinitely. This is primarily for two reasons. First of all, we expect the stratification in performance with m to continue until convergence at large M; increasing m for a given M increases errors. Second, the BCME model loses numerical stability at large m and M. Recall that the inverse of the global covariance matrix includes a term that is the sum of m matrices of size $M \times M$ (Equation 2.23b). Increasing m and M increases the chances of this matrix having a large condition number, which can then ruin the stability of predictions. Therefore, there is an upper limit in M beyond which the BCME method breaks down for a given value of m. We found empirically that this limit was at $M/N^G = 12\%$ for SARCOS using $m = 100$ and 200.

2.4 Discussion

The BCME provides competitive accuracy with highly attractive computational costs compared to popular sparse GP methods. Nevertheless, the method is not perfect. We already anticipated the tradeoff between accuracy and robustness to nodal failure (i.e. smaller m is more accurate but less robust; Equation 2.21's approximations are more easily satisfied with clustered data, but randomly distributing data is more robust). However, we now see an additional tradeoff between accuracy and cost: decreasing m can improve performance at the expense of longer hyperparameter training times. Additionally, the BCME has intrinsic performance limits: we can increase M only so far before the model breaks down and loses numerical stability. We will consider techniques to ameliorate the magnitudes of these tradeoffs in Chapter 4. Overall, the BCME method has great promise. The empirical results we have shown are otherwise unprecedented in term of the accuracy we have achieved given the training times we have measured.



(a) $SMSE$, Abalone

(b) $SNLP$, Abalone

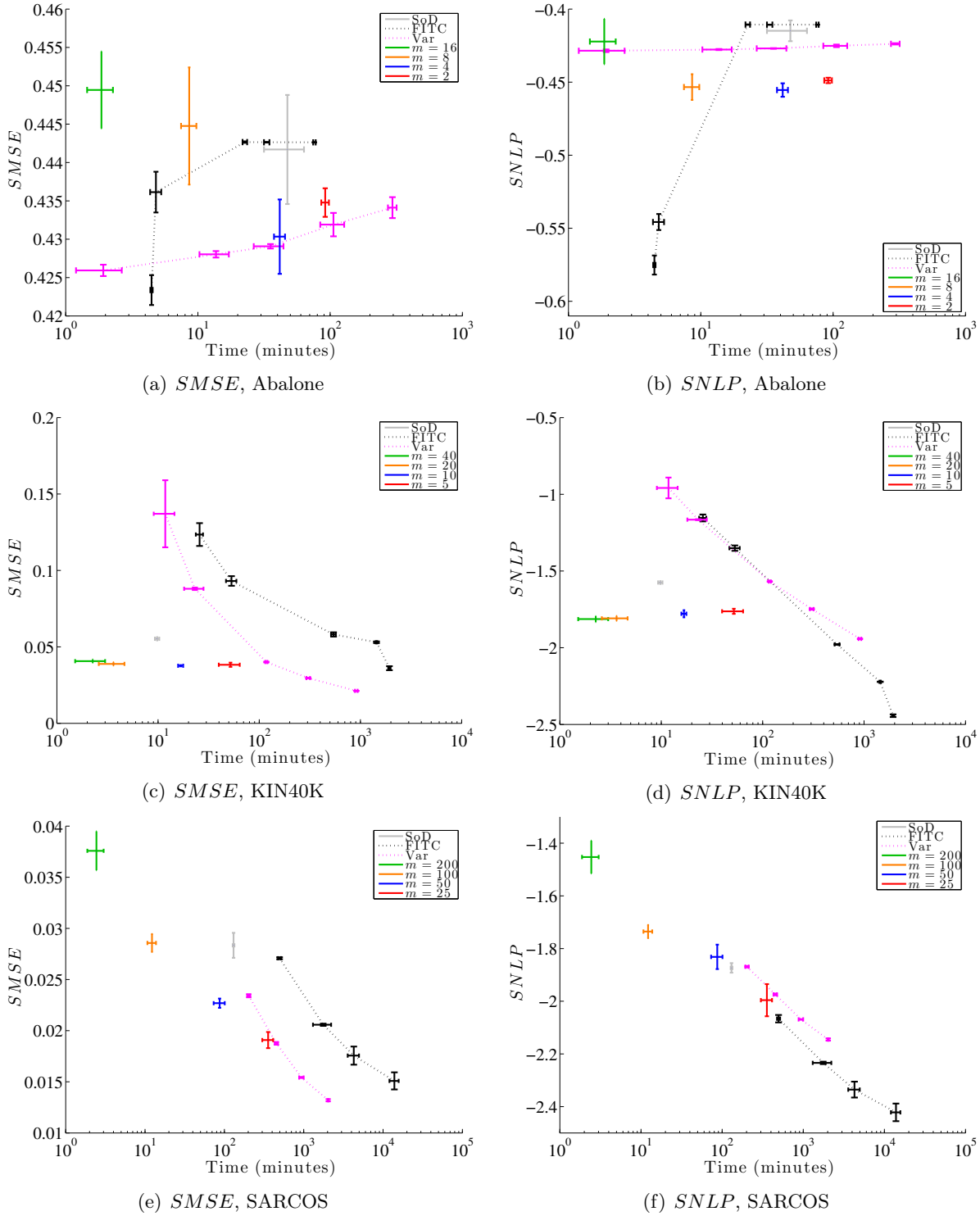
(c) $SMSE$, KIN40K

(d) $SNLP$, KIN40K

(e) $SMSE$, SARCOS

(f) $SNLP$, SARCOS

Figure 2.6. Error measures vs. theoretical hyperparameter training time for all GP methods. Vertical error bars extend ± one standard deviation from the mean value over five trials. Horizontal error bars for the BCME models indicate the mean training time ± one standard deviation over the five trials, which result from varying cluster sizes as we repeat experiments. For each BCME network size (i.e. m) we show only the results with the largest M used for experimentation. FITC and Var are shown for all M values tested; larger times and smaller errors correspond to larger M.



(a) $SMSE$, Abalone

(b) $SNLP$, Abalone

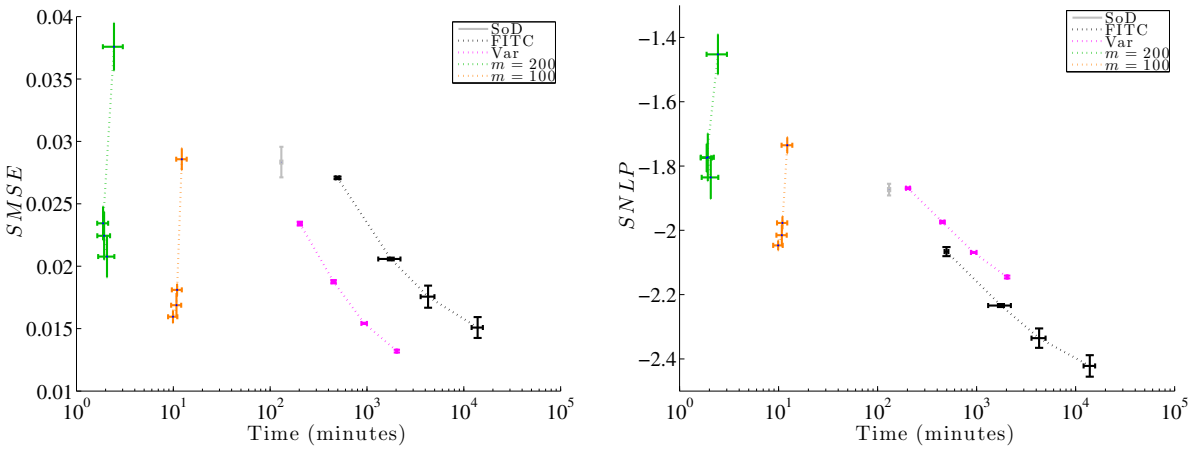
(c) $SMSE$, KIN40K

(d) $SNLP$, KIN40K

(e) $SMSE$, SARCOS

(f) $SNLP$, SARCOS

Figure 2.7. Error measures vs. empirical hyperparameter training time for all GP methods. Vertical error bars extend ± one standard deviation from the mean value over five trials. Horizontal error bars indicate the mean training time ± one standard deviation over the five trials, which result from both varying cluster sizes for the BCME models as well as other practical factors such as server loads for all models while running experiments. For each BCME network size (i.e. m) we show only the results with the largest M used for experimentation. FITC and Var are shown for all M values tested; larger times and smaller errors correspond to larger M.



(a) $SMSE$, SARCOS (b) $SNLP$, SARCOS

Figure 2.8. Error measures vs. empirical hyperparameter training time with large m and M. The values for FITC and Var are the same as in Figure 2.7(e) and 2.7(f). For the BCME, we plot only the cases of $m = 100$ and 200, and we show $M/N^G = 4\%$ (as in Figure 2.7) as well as 8%, 10% and 12%. Vertical and horizontal error bars have the same meaning as in Figure 2.7. For all models, larger M corresponds to smaller errors. For FITC and Var, larger M also corresponds to longer times, but this is not the case for the BCME models with a given m.

Chapter 3

Moment Matching via Message Passing

Motivation

In the PITC/BCM/BCME approximations to Gaussian process regression presented in Chapter 2, latent function values are considered to be composed of conditionally independent blocks given inducing inputs, which is equivalent to having a block diagonal structure for the conditional distribution $P(\mathbf{f}|\mathbf{u}, X, X_u)$. In the BCM/BCME representations, this block diagonal structure is physically explicit: training data are distributed amongst m nodes in a network, and each node's data corresponds to a block in the conditional covariance matrix.

When a node fails, the network loses all of the data that was in that node. A network is robust to such nodal failure when its predictive power is virtually unaffected by this loss of data, regardless of the points at which we choose to perform predictions. Note that inducing variables can only decrease the conditional dependence between latent function values (the original covariance matrix between data is subtracted by the positive semidefinite matrix Q_{ff}) [13]. In other words, it is possible to choose inducing variables poorly so that they do not decrease the dependence between blocks of latent function values, but data that are unconditionally independent will remain so for any choice of inducing inputs. Therefore, a robust system requires that all m nodes have similar training data, i.e. data that is unconditionally extremely dependent between nodes. A "perfectly robust" system would have data that is distributed uniformly at random amongst all nodes. Again, we make note of the interplay between accuracy and robustness. Whereas the assumptions of our BCME regression model are more easily satisfied when the data is clustered, the system is most robust to nodal failure when each node has a random subset of training data.

In some physical systems such as wireless sensor networks, randomly distributing data from a central authority is not always realistic. Rather, each node collects its own data, and this data could characterize part of a system that other nodes cannot observe due to the physically distributed nature of the network. In this case, increasing robustness of the system can occur via exchanges of data. One method to increase the "similarity" between datasets is to match moments. We assume that the m nodes can each only store N data points, so they must must exchange, rather than just transfer, data with each other.[1] The stopping criterion for the message-passing process depends on the probabilities of failure (or equivalently the expected lifetimes) of the nodes. A node that has a high probability of failure will want to ensure that it gives as much information as possible to the other nodes before dying. On the other hand, nodes that have

[1] In this chapter, we will consider each node to have the same number of data points: $N^i = N$ and $N^G = mN$.

infinite lifetimes do not care about trying to increase dependence between datasets; robustness to nodal failure is not an issue for these systems.

It is reasonable to consider the possibility of exchanging inducing input/output pairs, since (by assumption) the inducing variables contain a compressed representation of the data in each node. However, this would require that each node first train a GP regression model. When nodes are prone to failure, we would rather perform the moment-matching process before performing regression, since, as we will show, the moment-matching process is much more efficient than GP regression. This approach minimizes the impact of nodal failure (whether it occurs during moment matching or regression) on predictions.

3.1 An Asynchronous Gossip Algorithm for Message Passing over Directed Graphs

For the moment, we make the following assumptions (all of which will be relaxed later): the training inputs are scalar, the nodes only want to match the means of their training inputs, and in each node there always exists a subset of data of any size whose mean matches the mean of its N data points.

Intuition for the algorithm Consider a *synchronous* process where at every time step, node i exchanges a subset of its data with each of its neighbors; this subset has the same mean as the current distribution of data in node i, and similarly, the data that node i receives from node j has the same mean as the data in node j. Then, the dynamics for the mean of data in node i, $x_i(t) \in \mathbb{R}$, are:

$$x_i(t+1) = x_i(t) - s\left(x_i(t) - \sum_{j \in \mathcal{N}_i} a_{ij} x_j(t)\right), \tag{3.1a}$$

$$a_{ij} \geq 0, \quad \sum_{j \in \mathcal{N}_i} a_{ij} = 1, \tag{3.1b}$$

where $\mathcal{N}_i$ indicates the set of neighbors of node i. The a_{ij} are fractional weights indicating the relative amounts with which node i interacts with each of its neighbors, and $s = c/N$, where c is the total amount of data that node i exchanges with another node. Now suppose that all nodes always share the same amount of data at each step (s is fixed for all nodes and time). Collecting the mean of all nodes in $\mathbf{x}(t) \in \mathbb{R}^m$, the system dynamics are:

$$\mathbf{x}(t+1) = (I - sL)\mathbf{x}(t), \tag{3.2a}$$

$$L = I - A, \quad [A]_{ij} = a_{ij} \tag{3.2b}$$

L is the (normalized) Laplacian for the graph defining the communication constraints of the network. It is a well known result that the means of all nodes are guaranteed to converge asymptotically (i.e. L is Hurwitz) as long as the graph is connected and $s < 1$ (see, for example, [11, 26] for illuminating discussions on graph Laplacians related to consensus dynamics).

The synchronous algorithm above requires a strong level of communication between all nodes throughout time, which can be demanding for a network. We now present an asynchronous variant, a "gossip"-style algorithm based on those presented in [2] wherein each node only interacts with a single neighbor at any given time. The nodes are modeled as unit rate Poison processes,

so the overall graph is a Poisson process with rate m. Let node i's clock tick at the k-th time slot. Node i contacts node j with probability p_{ij}, where $p_{ij} > 0 \iff j \in \mathcal{N}_i$,[2] and they exchange mean-centered data with each other as in the previous section. Thus, the dynamics for the system in the k-th time step are:

$$\mathbf{x}(k) = W_s(k)\mathbf{x}(k-1), \tag{3.3}$$

where $s \in (0,1)$ behaves exactly as in the synchronous version above to indicate the amount of data exchanged relative to N, the total amount of data in each node. With probability p_{ij}/m (the $1/m$ factor is to select node i), the matrix $W_s(k)$ is:

$$W_s^{ij} = I - s\left(\mathbf{e}_i - \mathbf{e}_j\right)\left(\mathbf{e}_i - \mathbf{e}_j\right)^T, \tag{3.4}$$

where $\mathbf{e}_i$ is the unit vector with the i-th element equal to 1.[3]

3.1.1 Convergence in Expectation

The expected dynamics of $\mathbf{x}(t), t \geq 1$ are given by:

$$E\left[\mathbf{x}(t)\right] = E\left[W_s(t)W_s(t-1)...W_s(1)\mathbf{x}(0)\right] = \prod_{i=1}^{t} E\left[W_s(i)\right]\mathbf{x}(0) = \overline{W_s}^t\mathbf{x}(0), \tag{3.5}$$

where $\overline{W_s} := E[W_s(i)]$. To guarantee convergence (in expectation) to the mean of the entire dataset for any initial conditions, we would like to show that $\lim_{t\to\infty}\overline{W_s}^t = \mathbf{1}\mathbf{1}^T/m$. We can compute $\overline{W_s}$ as follows:

$$\overline{W_s} = \sum_{i,j}\left(\frac{1}{m}p_{ij}\right)W_s^{ij} = I - \frac{s}{m}\hat{L} \tag{3.6a}$$

$$\hat{L} = D - \hat{A}, \;\; \hat{A} = P + P^T, \;\; [P]_{ij} = p_{ij}, \;\; [D]_{ii} = \sum_{i=1}^{m} p_{ij} + p_{ji}. \tag{3.6b}$$

Note that, for symmetric P, we have $D = 2I$, so $\overline{W_s} = I - \frac{2s}{m}(I - P)$.

To begin analysis, we restrict $s \geq 0$, as negative s is not physically meaningful. For $0 \leq s \leq 1$, $\overline{W}_s$ is the weighted average of nonnegative, doubly stochastic matrices, so it is also a nonnegative, doubly stochastic matrix. Furthermore, we can consider P to be an adjacency matrix for a directed graph. As long as P is connected (there exists a globally reachable node), $\hat{A}$ is the adjacency matrix for a strongly connected graph (all nodes can reach each other through directed edges). Then, $\hat{L}$ is the Laplacian for a strongly connected graph and $\overline{W}_s$ is irreducible for $s > 0$. This implies that 1 is a simple eigenvalue for $s > 0$.

Additionally, we can show that $\overline{W_s}$ is a primitive irreducible stochastic matrix for $0 < s < 1$, i.e. it has only one eigenvalue with maximum modulus. The maximum modulus eigenvalue for a stochastic matrix has a modulus of 1. Furthermore, a doubly stochastic matrix has only real eigenvalues since it is symmetric. Thus, to be primitive, $\overline{W_s}$ cannot have -1 as an eigenvalue. Because $\max\sum_{j\neq i}[\hat{A}]_{ij} = m$ for any graph, the Gershgorin Circle Theorem guarantees that all eigenvalues for $\overline{W_s}$ are contained in the range $[1-2s, 1]$. Thus, $\overline{W_s}$ is primitive for $s \in (0,1)$.

[2] We do not consider self-loop edges, so $i \notin \mathcal{N}_i$ and $p_{ii} = 0$.

[3] In [2], s is fixed at $1/2$ and all of the assumptions we mentioned at the beginning of this section are retained. Our results will be more general in that we have $s \in (0,1)$ and we will systematically remove all of the aforementioned assumptions.

To summarize, we have that $\overline{W}_s$ is a primitive doubly stochastic matrix for $s \in (0,1)$ and P encoding a connected graph topology. Convergence in expectation follows from the Perron-Frobenius Theorem:

Theorem 3.1. (Perron-Frobenius Theorem, [7]) *Let Y be a primitive nonnegative matrix with eigenvectors satisfying $Y^T\mathbf{u} = \mathbf{u}$, $Y\mathbf{v} = \mathbf{v}$, and $\mathbf{v}^T\mathbf{u} = 1$. Then, $\lim_{t\to\infty} Y^t = \mathbf{v}\mathbf{u}^T$.*

Because $\overline{W_s}$ is doubly stochastic, $\mathbf{u} = \mathbf{v} = \mathbf{1}/\sqrt{m}$. Therefore, $\lim_{t\to\infty} \overline{W_s}^t = \mathbf{1}\mathbf{1}^T/m$.

3.1.2 Convergence Speed

The convergence speed is computed in [2] by bounding the dynamics of the second moment. We now show a similar result.[4] The average value to which the system converges ($\bar{x} = \mathbf{1}^T\mathbf{x}(0)/m$) is invariant under the dynamics:

$$\mathbf{1}^T\mathbf{x}(k) = \mathbf{1}^T W_s(k)\mathbf{x}(k-1) = \mathbf{1}^T\mathbf{x}(k-1) = m\bar{x} \tag{3.7}$$

Denote the dispersion vector as $\mathbf{z}(k) := \mathbf{x}(k) - \bar{x}\mathbf{1}$. This vector follows the same dynamics as $\mathbf{x}$:

$$\begin{aligned}\mathbf{z}(k) &= \mathbf{x}(k) - \bar{x}\mathbf{1} \\ &= W_s(k)\mathbf{x}(k-1) - W_s(k)\bar{x}\mathbf{1} \\ &= W_s(k)\mathbf{z}(k-1).\end{aligned} \tag{3.8}$$

Also, $\mathbf{z} \perp \mathbf{1}$:

$$\mathbf{1}^T\mathbf{z}(k) = \mathbf{1}^T\mathbf{x}(k) - \bar{x}\mathbf{1}^T\mathbf{1} = m\bar{x} - m\bar{x} = 0. \tag{3.9}$$

Next, note that

$$(W_s^{ij})^T W_s^{ij} = \left(I - s\left(\mathbf{e}_i - \mathbf{e}_j\right)\left(\mathbf{e}_i - \mathbf{e}_j\right)^T\right)^2 = W_r^{ij}, \;\; r = 2s(1-s). \tag{3.10}$$

Then, $E[W_s(k)^T W_s(k)] = E[W_r(k)] = \overline{W_r}$ and

$$\begin{aligned}E\left[\mathbf{z}(k+1)^T\mathbf{z}(k+1)\right] &= E\left[\mathbf{z}(k)^T W_s(k+1)^T W_s(k+1)\mathbf{z}(k)\right] \\ &= E\left[\mathbf{z}(k)^T E\left[W_s(k+1)^T W_s(k+1)|\mathbf{z}(k)\right]\mathbf{z}(k)\right] \\ &= E\left[\mathbf{z}(k)^T\overline{W_r}\mathbf{z}(k)\right],\end{aligned} \tag{3.11}$$

where we have used the fact that W_s is independent of $\mathbf{z}$. For $s \in (0,1)$, $r \in (0, 1/2]$, so the eigenvalues of $\overline{W_r}$ are guaranteed to be in the range $[0,1]$, and $\overline{W_r}$ is positive-semidefinite. Since $\mathbf{z} \perp \mathbf{1}$, the following must be be true:

$$\mathbf{z}^T(k)\overline{W_r}\mathbf{z}(k) \leq \lambda_2(\overline{W_r})\mathbf{z}^T(k)\mathbf{z}(k), \tag{3.12}$$

where $\lambda_2(\overline{W_r}) < 1$ is the second largest eigenvalue of $\overline{W_r}$. Then,

$$E\left[\mathbf{z}(k+1)^T\mathbf{z}(k+1)\right] \leq \lambda_2(\overline{W_r})E\left[\mathbf{z}^T(k)\mathbf{z}(k)\right], \tag{3.13}$$

[4] Note that the speed of convergence is currently defined as the rate at which the means of all m nodes reach the mean of the original dataset ($\bar{x} = \mathbf{1}^T\mathbf{x}(0)/m$). Later, we will analyze the situation where we only care about the convergence of the mean of a single node, the leader of the group, to $\bar{x}$.

so $E\left[\mathbf{z}(k)^T\mathbf{z}(k)\right] \leq \lambda_2(\overline{W_r})^k\mathbf{z}^T(0)\mathbf{z}(0)$. Since $\mathbf{z}^T(0)\mathbf{z}(0) = \mathbf{x}^T(0)\mathbf{x}(0) - m\bar{x}^2 \leq \mathbf{x}^T(0)\mathbf{x}(0)$, we can use Markov's inequality to bound the convergence time[5]:

$$\begin{aligned} P\left(\frac{\|\mathbf{z}(k)\|}{\|\mathbf{x}(0)\|} \geq \epsilon\right) &= P\left(\frac{\mathbf{z}(k)^T\mathbf{z}(k)}{\mathbf{x}(0)^T\mathbf{x}(0)} \geq \epsilon^2\right) \\ &\leq \epsilon^{-2}\frac{E\left[\mathbf{z}(k)^T\mathbf{z}(k)\right]}{\mathbf{x}(0)^T\mathbf{x}(0)} \\ &\leq \epsilon^{-2}\lambda_2(\overline{W_r})^k. \end{aligned} \tag{3.14}$$

Then we have $P\left(\|\mathbf{z}(k)\|/\|\mathbf{x}(0)\| \geq \epsilon\right) \leq \epsilon$ for $k \geq 3\log\epsilon/\log\lambda_2(\overline{W_r})$. In this way, we define the critical number of iterations needed for convergence as $k^* = \left\lceil 3\log\epsilon/\log\lambda_2(\overline{W_r})\right\rceil$. For a given system, we choose ϵ based on the probabilities of failure of the nodes. Systems with many nodes highly susceptible to failure will prefer smaller ϵ (even if they cannot finish k^* iterations before the nodes die), whereas networks with nodes resistant to failure can set $\epsilon = 1$.

3.1.3 Incorporating Noise

Now we remove the assumption that there exists a subset of data of any size whose mean matches the mean of the all N data points in a node. Instead, each node picks a subset whose mean is close to the true mean but corrupted by noise:

$$\mathbf{x}(k) = W_s(k)\mathbf{x}(k-1) + \boldsymbol{\xi}(k). \tag{3.15}$$

With probability p_{ij}/m, $\boldsymbol{\xi}(k)$ is given by:

$$\boldsymbol{\xi}^{ij}(k) = M^{ij}\mathbf{n}(k), \quad M^{ij} = -\left(\mathbf{e}_i - \mathbf{e}_j\right)\left(\mathbf{e}_i - \mathbf{e}_j\right)^T, \tag{3.16a}$$

$$E[\mathbf{n}(k)] = 0, \quad E[\mathbf{n}(k)\mathbf{n}^T(k)] = \sigma^2 I. \tag{3.16b}$$

Each component of the noise $\mathbf{n}(k)$ in Equation 3.16b depends only on the distribution of data within a particular node at iteration k.[6] In particular, this means that the noise is independent of other quantities such as $\mathbf{z}$ or W_s, and each node's noise is independent of the noise in all other nodes during that iteration.

Convergence in expectation still holds, because

$$\begin{aligned} E\left[\mathbf{x}(t)\right] &= E\left[W_s(t)\mathbf{x}(t-1)\right] + E[\boldsymbol{\xi}(t)] \\ &= \overline{W_s}E[\mathbf{x}(t-1)] + \sum_{i,j}\left(\frac{1}{m}p_{ij}\right)M^{ij}E[\mathbf{n}(t)] \\ &= \overline{W_s}E[\mathbf{x}(t-1)] \\ &= \overline{W_s}^t\mathbf{x}(0). \end{aligned} \tag{3.17}$$

However, the presence of noise changes the expression for convergence speed. The original mean

[5] We can also use $\|\mathbf{z}(0)\|$ as the normalizing constant, but we use $\|\mathbf{x}(0)\|$ to be consistent with the presentation in [2]. Either choice gives exactly the same result, although using $\|\mathbf{z}(0)\|$ is stronger, since $\|\mathbf{z}(0)\| \leq \|\mathbf{x}(0)\|$.

[6] We have omitted explicit conditioning on the data for conciseness. All probabilistic expressions will be conditioned on the data distribution across all nodes, unless we explicitly state otherwise.

$\bar{x}$ is still invariant under the new dynamics:

$$\mathbf{1}^T\mathbf{x}(k) = \mathbf{1}^T W_s(k)\mathbf{x}(k-1) + \mathbf{1}^T\boldsymbol{\xi}(k) = \mathbf{1}^T\mathbf{x}(k-1) + 0 = m\bar{x}. \tag{3.18}$$

Also, $\mathbf{z}$ has the same dynamics as $\mathbf{x}$ and is orthogonal to the converged state:

$$\begin{aligned}\mathbf{z}(k) &= \mathbf{x}(k) - \bar{x}\mathbf{1} \\ &= W_s(k)\mathbf{x}(k-1) + \boldsymbol{\xi}(k) - W_s(k)\bar{x}\mathbf{1} \\ &= W_s(k)\mathbf{z}(k-1) + \boldsymbol{\xi}(k).\end{aligned} \tag{3.19a}$$

$$\mathbf{1}^T\mathbf{z}(k) = \mathbf{1}^T\mathbf{x}(k) - \bar{x}\mathbf{1}^T\mathbf{1} = m\bar{x} - m\bar{x} = 0. \tag{3.19b}$$

The main difference is in the second moment:

$$\begin{aligned}E\left[\mathbf{z}(k+1)^T\mathbf{z}(k+1)\right] &= E\left[\mathbf{z}(k)^T\overline{W_r}\mathbf{z}(k)\right] + E[\mathbf{z}(k)^T W_s(k+1)^T\boldsymbol{\xi}(k+1)] \\ &\quad + E[\boldsymbol{\xi}(k+1)^T W_s(k+1)\mathbf{z}(k)] + E[\boldsymbol{\xi}(k+1)^T\boldsymbol{\xi}(k+1)] \\ &= E\left[\mathbf{z}(k)^T\overline{W_r}\mathbf{z}(k)\right] + E[\boldsymbol{\xi}(k+1)^T\boldsymbol{\xi}(k+1)],\end{aligned} \tag{3.20}$$

where we have used the fact that $\mathbf{n}$ is independent of everything else to eliminate the cross-terms between $\mathbf{z}(k)$ and $\boldsymbol{\xi}(k+1)$.

The autocorrelation of the noise vector is:

$$\begin{aligned}E[\boldsymbol{\xi}(k)^T\boldsymbol{\xi}(k)] &= \sum_{i,j}\left(\frac{1}{m}p_{ij}\right)E\left[\mathbf{n}(k)^T(M^{ij})^T M^{ij}\mathbf{n}(k)\right] \\ &= \sum_{i,j}\left(\frac{1}{m}p_{ij}\right)E\left[-2\mathbf{n}(k)^T M^{ij}\mathbf{n}(k)\right] \\ &= \sum_{i,j}\left(\frac{1}{m}p_{ij}\right)(4\sigma^2) = 4\sigma^2.\end{aligned} \tag{3.21}$$

Then $E\left[\mathbf{z}(k+1)^T\mathbf{z}(k+1)\right] \leq \lambda_2(\overline{W_r})E\left[\mathbf{z}^T(k)\mathbf{z}(k)\right] + 4\sigma^2$, whereby

$$E\left[\mathbf{z}(k)^T\mathbf{z}(k)\right] \leq \lambda_2(\overline{W_r})^k\mathbf{z}^T(0)\mathbf{z}(0) + 4\sigma^2\frac{1-\lambda_2(\overline{W_r})^k}{1-\lambda_2(\overline{W_r})}, \tag{3.22}$$

Using Markov's inequality as before, we can get an upper bound on the convergence time. However, Markov's inequality only applies to nonnegative random variables, so we must first add a technical condition that the dispersion at time k is at least as large as the envelope of noise corruption. Specifically, define the condition Q as follows:

$$Q(k) := \mathbf{z}(k)^T\mathbf{z}(k) \geq 4\sigma^2\frac{1-\lambda_2(\overline{W_r})^k}{1-\lambda_2(\overline{W_r})} \tag{3.23}$$

The envelope of noise corruption grows and approaches a constant value at large k. Conditioning

on the event Q yields the same results as above:

$$\begin{aligned} E\left[\mathbf{z}(k+1)^T\mathbf{z}(k+1)|Q(k+1)\right] &= E\left[\mathbf{z}(k)^T\overline{W_r}\mathbf{z}(k)|Q(k+1)\right] + E[\boldsymbol{\xi}(k+1)^T\boldsymbol{\xi}(k+1)|Q(k+1)] \\ &\leq \lambda_2(\overline{W_r})E\left[\mathbf{z}^T(k)\mathbf{z}(k)|Q(k+1)\right] + 4\sigma^2 \\ E\left[\mathbf{z}(k)^T\mathbf{z}(k)|Q(k)\right] &\leq \lambda_2(\overline{W_r})^k\mathbf{z}^T(0)\mathbf{z}(0) + 4\sigma^2\frac{1-\lambda_2(\overline{W_r})^k}{1-\lambda_2(\overline{W_r})} \end{aligned} \tag{3.24}$$

Finally, we can apply Markov's inequality conditioned on Q:

$$\hat{\epsilon}(k) := \sqrt{\epsilon^2 + \frac{4\sigma^2}{\mathbf{x}(0)^T\mathbf{x}(0)}\frac{1-\lambda_2(\overline{W_r})^k}{1-\lambda_2(\overline{W_r})}} \tag{3.25a}$$

$$\begin{aligned} P\left(\frac{\|\mathbf{z}(k)\|}{\|\mathbf{x}(0)\|} \geq \hat{\epsilon}(k)\middle|Q(k)\right) &= P\left(\frac{\mathbf{z}(k)^T\mathbf{z}(k)}{\mathbf{x}(0)^T\mathbf{x}(0)} \geq \epsilon^2 + \frac{4\sigma^2}{\mathbf{x}(0)^T\mathbf{x}(0)}\frac{1-\lambda_2(\overline{W_r})^k}{1-\lambda_2(\overline{W_r})}\middle|Q(k)\right) \\ &= P\left(\frac{\mathbf{z}(k)^T\mathbf{z}(k)}{\mathbf{x}(0)^T\mathbf{x}(0)} - \frac{4\sigma^2}{\mathbf{x}(0)^T\mathbf{x}(0)}\frac{1-\lambda_2(\overline{W_r})^k}{1-\lambda_2(\overline{W_r})} \geq \epsilon^2\middle|Q(k)\right) \\ &\leq \epsilon^{-2}\left(\frac{E\left[\mathbf{z}(k)^T\mathbf{z}(k)|Q(k)\right]}{\mathbf{x}(0)^T\mathbf{x}(0)} - \frac{4\sigma^2}{\mathbf{x}(0)^T\mathbf{x}(0)}\frac{1-\lambda_2(\overline{W_r})^k}{1-\lambda_2(\overline{W_r})}\right) \\ &\leq \epsilon^{-2}\lambda_2(\overline{W_r})^k. \end{aligned} \tag{3.25b}$$

This expression is interpreted as finding the ϵ-bound of the dispersion (as in the case without noise) above the growing envelope of noise corruption. The asymptotic size of the envelope is the smallest we can ever guarantee for the asymptotic size of the dispersion (by setting $\epsilon \to 0$). Also, this expression holds only as long as the dispersion is larger than the size of the envelope (i.e. Q is true). For a given σ, as soon as the dispersion drops below the envelope, we no longer care about further convergence characteristics since the envelope only grows larger with more time steps. With this modification to what is considered convergence, k^* remains as before.

3.1.4 Multidimensional Data

Now we remove the assumption that the data is scalar, and each node's training data consists of d-dimensional vectors. In this way, $\mathbf{x}, \mathbf{z}, \boldsymbol{\xi}, \mathbf{n} \in \mathbb{R}^{md}$ and the new dynamics behave as if each dimension performs the moment-matching algorithm in parallel:

$$\mathbf{x}(k) = \left(W_s(k) \otimes I_d\right)\mathbf{x}(k-1) + \boldsymbol{\xi}(k), \tag{3.26a}$$

$$\boldsymbol{\xi}^{ij}(k) = \left(M^{ij} \otimes I_d\right)\mathbf{n}(k), \tag{3.26b}$$

$$E[\mathbf{n}(k)] = 0, \;\; E[\mathbf{n}(k)\mathbf{n}^T(k)] = I_m \otimes \Sigma \tag{3.26c}$$

$$\mathrm{diag}(\Sigma) = \left(\sigma_1^2, \sigma_2^2, ..., \sigma_d^2\right)^T, \tag{3.26d}$$

where I_d denotes the identity matrix $\in \mathbb{R}^{d\times d}$, the operation $\mathrm{diag}(A)$ forms a vector out of the diagonal elements of a matrix A, and σ_i denotes the autocorrelation of the noise for dimension i of the data. Importantly, the *dynamics* of convergence for all dimensions are uncoupled, even if the data of different dimensions are correlated. This is why we only need to specify the diagonal elements of Σ, as the off-diagonal terms are never used in the dynamics. Thus, instead of looking at the dispersion as the sum of dispersions in each dimension, we will look at each dimension

separately. Denote $\mathbf{x}_i$ and $\mathbf{z}_i$ as the mean and dispersion in the i-th dimension. The generalized convergence time cares about the probability that any one of the dimensions has a large dispersion. Denote R_i as the following event:

$$R_i(k) := \frac{\|\mathbf{z}_i(k)\|}{\|\mathbf{x}_i(0)\|} \geq \hat{\epsilon}_i(k) \tag{3.27a}$$

$$\hat{\epsilon}_i(k) := \sqrt{\epsilon^2 + \frac{4\sigma_i^2}{\mathbf{x}_i(0)^T\mathbf{x}_i(0)} \frac{1-\lambda_2(\overline{W_r})^k}{1-\lambda_2(\overline{W_r})}}. \tag{3.27b}$$

Also, define Q_i as the event that the dispersion in the i-th dimension is smaller than the corresponding noise envelope. Then, the convergence time can found using Markov's inequality as before:

$$\begin{aligned} P\left(\bigcup_{i=1}^{d} R_i(k) \middle| \bigcap_{i=1}^{d} Q_i(k)\right) &\leq \sum_{i=1}^{d} P\left(R_i(k) \middle| \bigcap_{i=1}^{d} Q_i(k)\right) \\ &\leq d\epsilon^{-2}\lambda_2(\overline{W_r})^k, \end{aligned} \tag{3.28a}$$

$$k \geq k^+ := \left\lceil \frac{3\log\epsilon - \log d}{\log \lambda_2(\overline{W_r})} \right\rceil \implies P\left(\bigcup_{i=1}^{d} R_i(k) \middle| \bigcap_{i=1}^{d} Q_i(k)\right) \leq \epsilon. \tag{3.28b}$$

Using Boole's inequality in Equation 3.28a is completely general; it makes no assumptions about the correlations between convergence of the different dimensions, and it crucially allows us to ignore off-diagonal terms in Σ.[7] The condition $\bigcap_{i=1}^{d} Q_i$ does not present a serious limitation on finding an upper bound for convergence time; if we know that dimension j is already almost converged (and that Q_j may fail very early in the moment-matching process), we can exclude this dimension and only track the remaining $d-1$ dimensions.

Note that if we were to compute instead the aggregate dispersion (the sum of dispersions in each dimension), the calculus of the previous section would yield k^* as the time for convergence, but with a noise envelope given by the sum of the individual noise envelopes (i.e. replacing $4\sigma^2$ with $4\sum_{i=1}^{d}\sigma_i^2$). However, if different dimensions of the data have drastically different magnitudes, this view allows the dimensions with larger magnitude to dominate: the non-convergence of dimensions with smaller magnitudes is hidden by the convergence of dimensions with larger magnitude. In this light, we will opt for the former view that treats each moment and dimension separately. This ensures that each dimension achieves convergence.

3.1.5 Matching Multiple Moments Simultaneously

So far, we have considered matching only the first moment (i.e the mean) of the data. To also match the second moment simultaneously, we can imagine appending each of the N data points with a d-dimensional vector of the element-wise squares of each value. For example, if the original datapoint was $(1,2,3)^T$, then the augmented datapoint becomes $(1,2,3,1,4,9)^T$. The same approach can be carried out for any moment we choose to match. Using this approach, the extra moments are treated analogously to extra dimensions in the data. Thus, to match v moments simultaneously, $\mathbf{x}, \mathbf{z}, \boldsymbol{\xi}, \mathbf{n} \in \mathbb{R}^{mdv}$. Denote $\mathbf{u}$ as a v-dimensional vector whose k-th element, u_k, is the value a moment we choose to match (i.e. 1 for the first moment or 2 for the

[7] If we used a higher-order Bonferroni inequality, which includes terms involving the intersections of R_i, we would need to know about the correlations between dimensions of the data.

second moment, etc.).[8] Then the dynamics are now given by:

$$\mathbf{x}(k) = (W_s(k) \otimes I_{dv})\, \mathbf{x}(k-1) + \boldsymbol{\xi}(k), \tag{3.29a}$$

$$\boldsymbol{\xi}^{ij}(k) = \left(M^{ij} \otimes I_{dv}\right) \mathbf{n}(k), \tag{3.29b}$$

$$E[\mathbf{n}(k)] = 0, \;\; E[\mathbf{n}(k)\mathbf{n}^T(k)] = I_m \otimes \Sigma \tag{3.29c}$$

$$\text{diag}(\Sigma) = \left(\sigma^2_{1u_1}, \sigma^2_{2u_1}, ..., \sigma^2_{du_1}, \sigma^2_{1u_2}, ..., \sigma^2_{du_v}\right)^T, \tag{3.29d}$$

where we now have autocorrelation values for each moment of each dimension. Denote $\mathbf{x}_{iu_j}$ and $\mathbf{z}_{iu_j}$ as the mean and dispersion in the i-th dimension and the u_j-th moment. The convergence time follows a similar pattern to that of the previous section, where we now look at the union of events over all d dimensions and v moments in each dimension. Denote the event R_{iu_j} as:

$$R_{iu_j}(k) := \frac{\|\mathbf{z}_{iu_j}(k)\|}{\|\mathbf{x}_{iu_j}(0)\|} \geq \hat{\epsilon}_{iu_j}(k) \tag{3.30a}$$

$$\hat{\epsilon}_{iu_j}(k) := \sqrt{\epsilon^2 + \frac{4\sigma^2_{iu_j}}{\mathbf{x}_{iu_j}(0)^T \mathbf{x}_{iu_j}(0)} \frac{1 - \lambda_2(\overline{W_r})^k}{1 - \lambda_2(\overline{W_r})}} \tag{3.30b}$$

and the corresponding Q_{iu_j} as:

$$Q_{iu_j}(k) := \mathbf{z}_{iu_j}(k)^T \mathbf{z}_{iu_j}(k) \geq 4\sigma^2_{iu_j} \frac{1 - \lambda_2(\overline{W_r})^k}{1 - \lambda_2(\overline{W_r})} \tag{3.30c}$$

Then the convergence criterion is given by $k^\dagger$:

$$k \geq k^\dagger := \left\lceil \frac{3\log\epsilon - \log d - \log v}{\log \lambda_2(\overline{W_r})} \right\rceil \implies P\left(\bigcup_{j=1}^{v} \bigcup_{i=1}^{d} R_{iu_j}(k) \,\middle|\, \bigcap_{j=1}^{v} \bigcap_{i=1}^{d} Q_{iu_j}(k) \right) \leq \epsilon. \tag{3.31}$$

As before, the use of Boole's inequality in the calculation of $k^\dagger$ allows us to sidestep the need to account for correlations between moments and dimensions of the data.

In the interest of keeping notation compact, we only considered matching higher moments for individual dimensions of the data. However, if we have prior knowledge that certain dimensions are highly correlated, we might prefer to check the convergence of a cross correlation between dimensions rather than the autocorrelations considered above. We can just as easily match cross correlations between dimensions in the same fashion of appending dimensions to the data.

Note that we can call the product dv the "effective" number of dimensions of the data that we wish to match. Because the convergence time scales with $\log \epsilon^3/dv$, the effective number of dimensions must be very large to make an appreciable impact on convergence time. For example, with $\epsilon = 0.1$, we would need $dv = 1000$ to multiply $k^\dagger$ by a factor of two compared to matching just one moment of scalar data.

Finally, we must point out another very crucial aspect of matching multiple moments. Above, we found that treating additional moments as extra dimensions essentially found a more conservative bound for the time required to match moments. To naively implement this formulation by actually

[8] This notation allows for the fact that we may want to match non-contiguous moments. For example, we may care about matching the first and third moments, in which case $u_1 = 1$ and $u_2 = 3$, or $\mathbf{u} = (1, 3)^T$.

appending the data with extra moments is impractical for memory-constrained systems, and it is unnecessary. By simply selecting a random subset of data of size sN, we draw from the underlying data distribution and therefore implicitly attempt to match all dimensions and moments. In this way, choosing a larger effective dimension size simply provides a more complete guarantee for convergence, or equivalently, a guarantee for convergence to a higher order.[9] The tradeoff in choosing random subsets of data is the increased noise level, which we analyze below.

3.1.6 Data Selection

In a single iteration of the moment-matching algorithm, a node must know its (vectorial) mean, which is composed of the means for all effective dimensions, and it must access sN data points with a mean as close as possible to this value. Keeping track and performing an online update for the mean when exchanging data is simple. All that is required is a variable holding the total (vectorial) sum of the data points, which is decremented and incremented with the loss and gain of data points respectively. As noted in the previous section, choosing points to match the mean can be done in various ways. We will first analyze a naive approach which stores data in a kd-tree, and we will then show a more elegant approach which involves choosing random subsets of data.

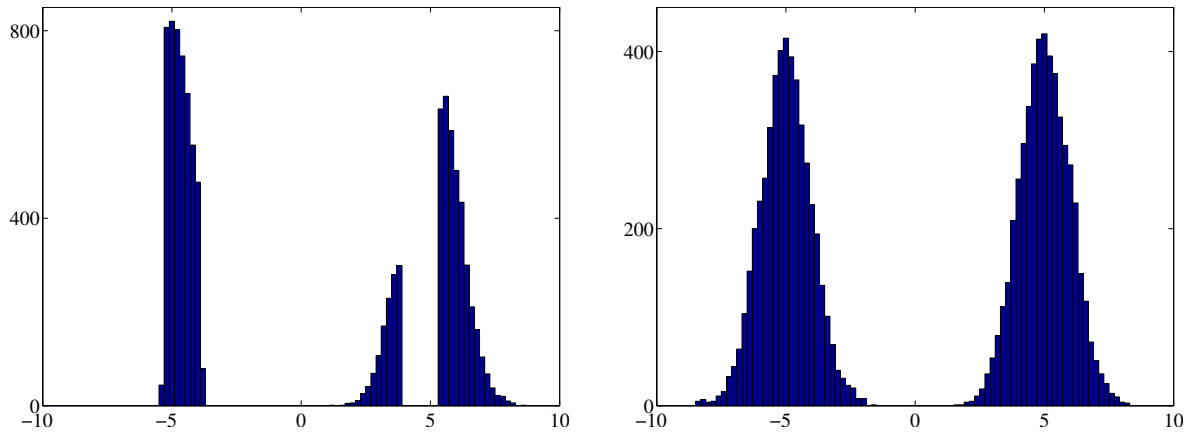
Naive Implementation Accessing the sN data points close to the mean value can be performed if the N data points are stored in a kd-tree and we search for sN nearest neighbors to the mean value. Such accesses, removals, and subsequent insertions of new points take $O(\log N)$ time each for a balanced kd-tree. Building and rebalancing a kd-tree each take $O(N \log N)$ time. If we assume that rebalancing only needs to occur after N additions and removals, we have an amortized $O(\log N)$ time complexity for rebalancing. To treat dimensions on an equal footing when we perform nearest neighbor search, we must use a distance metric that normalizes by a length scale. This can be done by normalizing the distance from a point to the mean by the standard deviation of that dimension.[10] This requires storage and update of an additional vector of the (vectorial) sum of squared data, which is used along with the vector of the sum of data points to compute the standard deviations.

This naive approach suffers from superlinear time complexity and it also requires appending data points with every moment we choose to match. This is clearly unacceptable from an efficiency standpoint, but this approach also suffers from a much more serious problem. Namely, failure to match higher-order moments creates "holes" in datasets that can ruin the ability for regression techniques to capture underlying trends. To illustrate this phenomenon, consider a two-node system in which the first node has 10^4 data points distributed according to $\mathcal{N}(5, 1)$, while the other has 10^4 data points drawn from $\mathcal{N}(-5, 1)$.[11] Suppose we perform moment matching for just the first moment. With $s = 1/4$ and $\epsilon = 0.1$, we have $k^\dagger = 5$ iterations. Figure 3.1(a) shows the data in the first node after running 5 iterations with data points selected by performing nearest neighbor search. The "holes" in the dataset are clearly undesirable, but to remove them would require matching higher-order moments. For the kd-tree implementation, this requires appending data points with the extra dimensions.

[9] Recall that $k^\dagger$ is interpreted as a guarantee for convergence time; the moments could have converged before $k^\dagger$ iterations.

[10] Performing nearest-neighbor search is still simple, as pruning of subtrees is done by checking the intersection of the splitting axis-aligned hyperplane with an axis-aligned hyperellipsoid, as opposed to a hypersphere for the non-normalized Euclidean distance metric. Because the ellipsoid is axis-aligned, this is still a simple operation. We can also view this as using Euclidean distance after first scaling the data, a.k.a Mahalanobis distance with a diagonal covariance matrix.

[11] $\mathcal{N}(m, v)$ indicates a normal distribution with mean m and variance v.



(a) Kd-tree implementation (b) Random subset implementation

Figure 3.1. Histograms of data in the first node after performing moment matching (shown with a bin width of 0.2). The random subset implementation is faster than the kd-tree implementation and automatically attempts to preserve the underlying shape of the aggregate distribution (which is equivalent to matching all moments).

Efficient Implementation A more efficient implementation is to simply exchange random subsets of data. Figure 3.1(b) shows the result after 5 iterations of exchanging random subsets. This method clearly leads to more desirable results in terms of retaining the underlying shape of the true distribution. It also has the distinct advantages of having a linear time complexity and no overhead for data storage.

3.1.7 Estimation of σ

Knowledge of the noise envelope is not necessary to perform the moment-matching process; it is only necessary if we wish to report the confidence bound to which we have converged after $k^\dagger$ iterations. When using random subsets for data selection, the sN data points randomly drawn from a node's dataset form a simple random sample without replacement. Thus, each entry of $\mathbf{n}$ is drawn from the distribution of the sample mean subtracted by the true mean, so σ simply represents the standard error of the sample mean.

As presented, the model for σ assumes a stationary noise distribution. Since the standard error of the sample mean depends on the variance of the underlying dataset, the assumption of stationarity is certainly not valid at the beginning of the moment-matching process, when a node's data could potentially change drastically with each iteration. Therefore, the easiest, yet least accurate, technique for estimating σ is the following: when each node accesses the sN data points, it also records the fluctuation of the mean of these data points from the requested mean. After the moment-matching process is completed, each node can then compute its own estimate of σ. Conservative estimates for each σ_{iu_j} can then be found by taking the maximum value of the standard error over all the nodes (which can be performed by flooding maximum values until all nodes have the same maximum). Finally, to get the value of the envelope, we also need to calculate $\|\mathbf{x}_{iu_j}(0)\|$, which can also be performed by flooding values until all nodes have the magnitudes from all other nodes. Even if the condition Q failed at some point of the moment-matching process, performing more iterations of moment matching only increases the magnitude of the convergence window (i.e. noise envelope), thereby providing a more conservative estimate of the degree of convergence.

A slightly more accurate technique has each node record fluctuations only from a subset of iterations towards the end of the moment-matching process. The motivation for this is twofold. First, the assumption that the noise level is stationary is more accurate towards the end of the moment-matching process. Second, if we were to compute the non-stationary noise envelope, the noise levels from early in the process would be multiplied by $\lambda_2(\overline{W_r})$ the most times in the computation, and therefore they would have less influence on the final value of the envelope compared to the noise levels towards the end of the process.

The most accurate technique is to remove the assumption of stationarity completely and track the growth of the non-stationary noise envelope with each iteration. However, this requires a degree of centralization which is unavailable in the current form of the gossip algorithm. Indeed, all nodes would need to have knowledge of the same global noise envelope, which adds an unacceptable communication overhead to the process.

3.1.8 Optimizing s

Optimization of s can involve either minimizing the amount of data passed during the moment-matching process ($k^\dagger sN$) or minimizing the time to reach convergence ($k^\dagger$). Considering the former, we have the following problem:

$$\min_{s\in(0,1)} k^+ sN \equiv \min_{s\in(0,1)} \frac{-s}{\log \lambda_2(\overline{W_r})} = \min_{s\in(0,1)} \frac{-s}{\log\left(1-\frac{2s(1-s)}{m}\lambda_{min}(\hat{L})\right)} := \min_{s\in(0,1)} F(s), \tag{3.32}$$

where $\lambda_{min}(\hat{L})$ is the smallest nonzero eigenvalue of $\hat{L}$. For large m, we can approximate the solution by the following argument. The largest value of $\lambda_{min}(\hat{L})$ is $2m/(m-1)$ which occurs for a homogeneous complete graph, where $p_{ij} = 1/(m-1)\ \forall\ i \neq j$ [18]. Thus, $\lambda_{min}(\hat{L})/m \leq 2/(m-1) = O(1/m)$. Therefore, $\lambda_{min}(\hat{L})/m \ll 1$ for large m, and $F(s)$ is approximately

$$F(s) \approx \frac{s}{\frac{2s(1-s)}{m}\lambda_{min}(\hat{L})} = \frac{m}{2\lambda_{min}(\hat{L})(1-s)}. \tag{3.33}$$

This expression is minimized when s is as small as physically possible: $s = 1/N$, which corresponds to exchanging one data point at each time step. Although this minimizes the amount of data shared, it results in very slow convergence speeds, since $\lambda_2(\overline{W_r}) \to 1$ as $s \to 0$.

Considering the second optimization problem, we have:

$$\arg\min_{s\in(0,1)} \frac{1}{s}F(s) = \arg\min_{s\in(0,1)} \frac{-1}{\log\left(1-\frac{2s(1-s)}{m}\lambda_{min}(\hat{L})\right)} = \frac{1}{2} \tag{3.34}$$

It appears as if convergence is quickest if nodes exchange half of their data with each other at every time step. However, this analysis only analyzes the number of iterations and does not take into account the overhead associated with each transaction. Particularly in large networks, bandwidth limitations can severely affect transmission rates and can push the optimal value of $s < 1/2$. Because the gossip algorithm's optimal s value is agnostic to the number of nodes in the system, it is not very scalable.

3.1.9 Optimizing $\lambda_2(\overline{W_r})$

Minimizing convergence time and the size of the noise corruption window requires minimizing $\lambda_2(\overline{W_r})$:

$$\begin{aligned} \min_P \ & \lambda_2(\overline{W_r}) \\ \text{subject to } & \overline{W_r} = \sum_{i,j} \frac{1}{m} p_{ij} W_r^{ij} \\ & p_{ij} \geq 0, \ \ p_{ij} = 0 \ \forall \ j \notin \mathcal{N}_i, \ \ \sum_j p_{ij} = 1 \ \forall \ i \end{aligned} \tag{3.35}$$

It is shown in [2] that this convex optimization problem can be solved in a decentralized manner using subgradient methods. As noted above, the best case possible is that of a homogeneous complete graph, in which $\lambda_2(\overline{W_r}) = 1 - 4s(1-s)/(m-1)$. A complete graph, however, is rarely achievable in realistic networks.

3.1.10 Shortcomings of the Model

We have so far only considered optimizing for speed without considering reliability of communication. Namely, the optimization in Section 3.1.9 chooses weights p_{ij} in a manner that is ignorant of the reliabilities of each node. It may not be optimal for a neighbor i of node j to have large p_{ij} if node j has an extremely short expected lifetime. Thus, speed and reliability must be considered simultaneously in a more sophisticated algorithm.

It is also necessary to reiterate that the optimization of Section 3.1.9 minimizes the speed of convergence as defined by *all nodes* having means close to the entire mean of the dataset. This robustness algorithm is most relevant to the case where data is collected in a distributed way, but only a single arbitrarily chosen node is queried for its predictions on a given test dataset. This could be the case, for example, when a prediction is needed very quickly and we do not want to wait for the nodes to combine their predictions. In the BCM/BCME model, however, we collect and combine the predictions of each node into a central leader node. In this sense, robustness is operationally defined as requiring the leader's data to look representative of the original dataset covering all nodes. Although the gossip-style algorithm will remain the same, convergence speeds can often be much faster than those governed by $\lambda_2(\overline{W_r})$. In addition, we will also find that the optimal s value will be more scalable to the number of nodes in the system. We consider this leader-centric gossip algorithm next.

3.2 A Leader-Centric Gossip Algorithm

The leader-centric gossip algorithm is similar to the completely decentralized version, because we still require nodes to only have N data points at a time, and nodes exchange data with a single neighbor at a given time. However, the speeds and the optimal graph structure (given by P) are different. In the leader-centric case, we only track how closely the leader's data resembles the overall distribution of data. In our moment-matching framework, we care only about the convergence characteristics of the leader's moments.

3.2.1 Convergence Speed for Restricted Leader-Centric Dispersion

In this subsection we briefly outline the intuition behind the fact that the convergence speed for a leader-centric algorithm is often faster than that for the original gossip algorithm.

Leader-Centric Dispersion

For the moment, assume that the input data is scalar. Let the leader be node 1, so that $\mathbf{e}_1^T\mathbf{z}(k) := z_1(k)$ is the leader's (scalar) dispersion at time k. Instead of just looking at this scalar, we will instead look at $\mathcal{D}$, the subspace of $\mathbb{R}^m$ governed by the eigenvectors of $\overline{W_r}$ with a nonzero component for the leader. Denote the m normalized eigenvectors of $\overline{W_r}$ as $\mathbf{v}_i,\ i \in \{1, 2, ..., m\}$ in descending order by eigenvalue. Then $\mathcal{D}$ and Π, the projection matrix onto $\mathcal{D}$, are defined by

$$\mathcal{D} = \text{span}\{\mathbf{v}_i : \mathbf{e}_1^T\mathbf{v}_i \neq 0, \mathbf{1}^T\mathbf{v}_i = 0\} \tag{3.36a}$$

$$\Pi = \sum_{i:\mathbf{v}_i\in\mathcal{D}} \mathbf{v}_i\mathbf{v}_i^T. \tag{3.36b}$$

Let $\mathbf{z}_L := \Pi\mathbf{z}$ be the "leader-centric" dispersion vector. The projected dynamics do not generally follow similar dynamics to those of $\mathbf{x}$ and $\mathbf{z}$, but the expected dynamics do:

$$\begin{aligned} E[\mathbf{z}_L(k)] &= \Pi E[\mathbf{z}(k)] \\ &= \Pi E[W_s(k)\mathbf{z}(k-1) + \boldsymbol{\xi}(k)] \\ &= \Pi\overline{W_s}E[\mathbf{z}(k-1)] \\ &= \overline{W_s}\Pi E[\mathbf{z}(k-1)] \\ &= \overline{W_s}E[\mathbf{z}_L(k-1)]. \end{aligned} \tag{3.37}$$

Since $\overline{W_s}$ is symmetric and therefore unitarily diagonalizable, all of the eigenvectors are orthogonal to each other. Thus, we know that $\overline{W_s}$ commutes with Π because Π is made up of its eigenvectors: this can be shown trivially using the definition of Π above and the fact that $\overline{W_s} = \sum_i^m \lambda_i\mathbf{v}_i\mathbf{v}_i^T$.

Restricted Leader-Centric Dispersion

Since $\mathcal{D}$ contains the dispersion of the leader, it seems reasonable that the convergence envelope for $\mathcal{D}$ should define the convergence envelope for z_1. This intuition is almost correct, but in general there are some qualifications. However, this intuition is exactly correct for a very restricted type of dispersion. Consider the restriction that $\mathbf{z}(k) \in \mathcal{D}\ \forall\ k$, or equivalently, $\mathbf{z}(k) = \mathbf{z}_L(k)\ \forall\ k$.

Then, the convergence speed can be found by computing the second moment as before:

$$\begin{aligned} E\left[\mathbf{z}(k+1)^T\mathbf{z}(k+1)|\mathbf{z}\in\mathcal{D}\right] &= E\left[\mathbf{z}(k)^T\overline{W_r}\mathbf{z}(k)|\mathbf{z}\in\mathcal{D}\right] + E[\boldsymbol{\xi}(k+1)^T\boldsymbol{\xi}(k+1)|\mathbf{z}\in\mathcal{D}] \\ &\leq \lambda_L(\overline{W_r})E\left[\mathbf{z}^T(k)\mathbf{z}(k)\right] + 4\sigma^2 \\ E\left[\mathbf{z}(k)^T\mathbf{z}(k)|\mathbf{z}\in\mathcal{D}\right] &\leq \lambda_L(\overline{W_r})^k\mathbf{z}^T(0)\mathbf{z}(0) + 4\sigma^2\frac{1-\lambda_L(\overline{W_r})^k}{1-\lambda_L(\overline{W_r})}, \end{aligned} \tag{3.38a}$$

where $\lambda_L(\overline{W_r})$ is given by:

$$\lambda_L(\overline{W_r}) = \max_i\{\lambda_i(\overline{W_r}) : \mathbf{v}_i \in \mathcal{D}\}. \tag{3.39}$$

In other words, $\lambda_L(\overline{W_r})$ is the largest eigenvalue associated with the subspace $\mathcal{D}$. To minimize convergence time, we would like to minimize $\lambda_L(\overline{W_r})$ instead of $\lambda_2(\overline{W_r})$ as we did for the "regular" gossip algorithm. Note that if $\mathcal{D}$ contains the eigenvector(s) associated with $\lambda_2(\overline{W_r})$, then $\lambda_2(\overline{W_r}) = \lambda_L(\overline{W_r})$. We are more interested in the opposite case where $\mathcal{D}$ is orthogonal to the subspace containing the slowest eigenmode. This case, however, results in a non-convex optimization problem, so we can not handle it in the same way as with the regular gossip algorithm.

3.2.2 The Efficiency of Symmetric Stars

An alternative to blindly attempting non-convex optimization for minimizing convergence time is to heuristically seek and characterize optimal graph structures. In this sense, we now characterize the properties of a symmetric star graph that render it optimal for our purposes.

When searching for an optimal graph structure, we mean that we are searching for a graph whose $\lambda_L(\overline{W_r})$ will be as small as possible. The best case scenario is to let $\lambda_L(\overline{W_r}) = \lambda_m(\overline{W_r})$, the smallest eigenvalue of $\overline{W_r}$. This corresponds to the the *largest* eigenvalue of the corresponding graph Laplacian, $\lambda_{max}(\hat{L})$ (which obviously has the same eigenvector). The largest eigenvalue of the graph Laplacian has been studied extensively. We now cite a particularly relevant bound (adapted to our notation):

Theorem 3.2. ([3]) *For a simple connected weighted graph, $\lambda_{max}(\hat{L})$ is bounded by:*

$$\lambda_{max}(\hat{L}) \leq \max_{i\sim j}\left\{[D]_{ii} + [D]_{jj}\right\}, \tag{3.40}$$

where $i \sim j$ indicates that i and j are neighbors. Equality holds if the graph is semiregular bipartite.

A simple graph is one with no self-loops and only one set of edges between vertices. It applies to our case because $\hat{L}$ is equivalent to the Laplacian of an undirected graph, even though P is directed and may have two edges between the same pair of vertices. A semiregular bipartite graph is a bipartite graph in which the nodes within a bipartition have the same (weighted) degree.

Corollary 3.3. *The maximum upper bound for $\lambda_{max}(\hat{L})$ is $m+2$.*

Proof. Computing $[D]_{ii} + [D]_{jj}$ is the same as finding the sum of the entries in the i-th and j-th rows and columns of P. The row sums are always constrained to be 1. The sum of the column sums attains a maximum of m when all nodes contact node i exclusively, node j exclusively, or both in some combination (nodes i and j can only look at each other since we do not allow self-loops). □

Importantly, this upper bound is over all graph configurations. As we are heuristically trying to find a graph structure that maximizes the largest Laplacian eigenvalue, we can search for a graph

whose maximum eigenvalue is close to this upper bound. A symmetric star is defined as a graph in which the leader is centrally located and connected to $m-1$ follower nodes. The follower nodes only look at the leader, and the leader looks equally at all followers: $p_{1j} = 1/(m-1)$, $p_{j1} = 1 \; \forall \; j \in \{2, 3, ..., m\}$.

Corollary 3.4. *For a symmetric star,* $\lambda_{max}(\hat{L}) = m^2/(m-1) = m + 2 - (m-2)/(m-1)$.

Proof. A symmetric star is semiregular bipartite. The leader has a weighted degree $[D]_{11} = m$, and all follower nodes have weighted degrees $[D]_{jj} = m/(m-1) \; \forall \; j \in \{2, 3, ..., m\}$. □

Therefore, the symmetric star's maximum Laplacian eigenvalue is just $(m-2)/(m-1)$ below the maximum attainable for any graph, so the relative or normalized difference is $\sim 1/m$ for large m, which is practically negligible.

We still need to characterize the eigenvectors of the symmetric star to find the convergence speed for restricted leader-centric dispersion. For $m > 2$, the Laplacian for a symmetric star has two distinct nonzero eigenvalues[12], $\lambda_A(\hat{L}) = m^2/(m-1)$ with multiplicity 1, and $\lambda_B(\hat{L}) = m/(m-1)$ with multiplicity $m-2$.[13] The corresponding orthogonal eigenvectors are:

$$\begin{aligned} \mathbf{v}_A(\hat{L}) &= \frac{(m-1)\mathbf{e}_1 - \sum_{i=2}^{m} \mathbf{e}_i}{\sqrt{m(m-1)}} \\ \mathbf{v}_{B_j}(\hat{L}) &= \frac{\sum_{i=2}^{j} \mathbf{e}_i - (j-1)\mathbf{e}_{j+1}}{\sqrt{j(j-1)}} \; \forall \; j \in \{2, 4, ..., m-1\}. \end{aligned} \tag{3.41}$$

Note that the only eigenvector touching the leader is $\mathbf{v}_A$ (excluding $\mathbf{1}$ which is orthogonal to the dispersion vector), so $\mathcal{D} = \text{span}\{\mathbf{v}_A\}$.

Asymmetry Ruins Efficiency At this point, it is reasonable to consider the case that the leader looks at the followers with different weights. This will increase the upper bound for $\lambda_{max}(\hat{L})$, so it could potentially increase the largest Laplacian eigenvalue compared to the case of the symmetric star. However, we only care about the largest Laplacian eigenvalue insofar as its corresponding eigenvector is the only eigenvector of $\hat{L}$ in $\mathcal{D}$. We now show that when the leader breaks symmetry by even a small amount, $\mathcal{D}$ picks up at least one more eigenvector with an eigenvalue much smaller than $m^2/(m-1)$.

An infinitesimal break in symmetry by the leader can be represented as a small perturbation δP to P, where δP has nonzero elements only in the first row, and the sum of the elements is 0. This perturbation must necessarily involve at least two follower nodes, and at least two follower nodes must have perturbations with opposite sign. Without loss of generality, let nodes 2 and 3 be involved in this opposing shift of resources by the leader (i.e. let $\delta p_{12} > 0$ and $\delta p_{13} < 0$ in the perturbation). Then, the corresponding perturbation to $\hat{L}$ will look like the following:

$$\delta\hat{L} = \begin{pmatrix} 0 & -\delta p_{12} & -\delta p_{13} & ... \\ -\delta p_{12} & \delta p_{12} & 0 & ... \\ -\delta p_{13} & 0 & \delta p_{13} & ... \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{3.42}$$

[12] Recall that 0 is a simple eigenvalue with eigenvector $\mathbf{1}$ because the graph is connected. For $m = 2$ the Laplacian has only two eigenvalues: 0 and λ_A.

[13] Because $\hat{L}$ is symmetric, the algebraic and geometric multiplicities of eigenvalues are equivalent.

Now, consider the effect of this perturbation to the eigenvector $\mathbf{v}_{B_2}$ (using a standard eigenvector perturbation formula for Hermitian matrices [24]):

$$\mathbf{e}_1^T \delta \mathbf{v}_{B_2} = \left(\frac{\mathbf{v}_A^T \delta \hat{L} \mathbf{v}_{B_2}}{\lambda_B - \lambda_A} \right) \mathbf{e}_1^T \mathbf{v}_A = \frac{\delta p_{12} - \delta p_{13}}{m\sqrt{2}} > 0. \tag{3.43}$$

Although the changes to the eigenvector are infinitesimal, the corresponding effects on convergence dynamics are drastic. Specifically, the leader-centric dispersion now contains at least (the perturbed form of) $\mathbf{v}_{B_2}$, and the perturbed eigenvalues remain close to their original values since the perturbation is small (an elementary fact given by the Bauer-Fike theorem or Weyl's matrix inequality). This brings the convergence speed down to a value close to $\lambda_2(\overline{W_r})$. Therefore, asymmetry by the leader ruins the efficiency of a star configuration.

Thus, we see that the symmetric star configuration satisfies our heuristic search for maximizing the speed of convergence for restricted leader-centric dispersions. When moving to the general case of non-restricted leader-centric dispersions, the intuition regarding the efficiency of symmetric stars is still relevant.

3.2.3 Convergence Speed for Unrestricted Leader-Centric Dispersion

Now we focus on unrestricted leader-centric dispersion for a symmetric star. Because the dynamics of $\mathbf{z}_L$ are not the same as those for $\mathbf{z}$ and $\mathbf{x}$, we will find that the benefits of the star configuration are corrupted by the slow convergence of the non-leader modes associated with $\mathcal{D}^\perp$, the subspace orthogonal to $\mathcal{D}$.

We will again assume that the input data are scalar for the moment and follow the same steps as before: use the recursive formula of an upper bound for the second moment of leader-centric dispersion to give an upper bound on convergence time. The second moment of leader-centric dispersion now looks like:

$$\begin{aligned} E\left[\mathbf{z}_L(k+1)^T \mathbf{z}_L(k+1)\right] &= E\left[\mathbf{z}(k+1)^T \Pi \mathbf{z}(k+1)\right] \\ &= E\left[\mathbf{z}(k)^T W_s(k+1)^T \Pi W_s(k+1) \mathbf{z}(k)\right] + E[\boldsymbol{\xi}(k+1)^T \Pi \boldsymbol{\xi}(k+1)] \\ &= E\left[\mathbf{z}(k)^T E\left[W_s(k+1)^T \Pi W_s(k+1) | \mathbf{z}(k)\right] \mathbf{z}(k)\right] \\ &\quad + E[\boldsymbol{\xi}(k+1)^T \Pi \boldsymbol{\xi}(k+1)]. \end{aligned} \tag{3.44}$$

As before, the inner expectation in the first term of the right hand side is independent of $\mathbf{z}(k)$. Denote $\overline{\Omega} := E\left[W_s(k)^T \Pi W_s(k)\right]$. Its computation requires some effort:

$$\begin{aligned} \overline{\Omega} &= \sum_{i,j} \left(\frac{1}{m} p_{ij} \right) W_s^{ij} \Pi W_s^{ij} \\ &= \sum_{i,j} \left(\frac{1}{m} p_{ij} \right) (I + sM^{ij}) \Pi (I + sM^{ij}) \\ &= \Pi \left(I + 2s\overline{M} \right) + \sum_{i,j} \left(\frac{1}{m} p_{ij} \right) s^2 M^{ij} \Pi M^{ij}, \end{aligned} \tag{3.45}$$

where we have used the fact that $\overline{W_s}$ commutes with Π iff $\overline{M}$ commutes with Π as well. The

second term can be expanded in the following way:

$$\begin{aligned}\sum_{i,j}\left(\frac{1}{m}p_{ij}\right)s^2M^{ij}\Pi M^{ij} &= \sum_{i,j}\left[\left(\frac{1}{m}p_{ij}\right)\left(-s^2M^{ij}\right)\sum_{k:\mathbf{v}_k\in\mathcal{D}}\left(\mathbf{e}_i-\mathbf{e}_j\right)^T\mathbf{v}_k\mathbf{v}_k^T\left(\mathbf{e}_i-\mathbf{e}_j\right)\right] \\ &= \sum_{i,j}\left[\left(\frac{1}{m}p_{ij}\right)\left(-s^2M^{ij}\right)\sum_{k:\mathbf{v}_k\in\mathcal{D}}\left(\left(\mathbf{e}_i-\mathbf{e}_j\right)^T\mathbf{v}_k\right)^2\right].\end{aligned} \tag{3.46}$$

Now, suppose that for all permutations of i and j with $p_{ij}\neq 0$,[14] the sum of the inner products is constant: $\sum_{k:\mathbf{v}_k\in\mathcal{D}}\left(\left(\mathbf{e}_i-\mathbf{e}_j\right)^T\mathbf{v}_k\right)^2=\alpha^2>0$.[15] Then, we arrive at an expression for $\overline{\Omega}$:

$$\begin{aligned}\overline{\Omega} &= \Pi\left(I+2s\overline{M}\right)-s^2\alpha^2\overline{M} \\ &= \Pi\left(I-2\frac{s}{m}\hat{L}\right)+s^2\frac{\alpha^2}{m}\hat{L},\end{aligned} \tag{3.47}$$

where we have used the fact that $\overline{M}=-\hat{L}/m$. A similar expression can be found for the noise autocorrelation:

$$\begin{aligned}E[\boldsymbol{\xi}(k)^T\Pi\boldsymbol{\xi}(k)] &= \sum_{i,j}\left(\frac{1}{m}p_{ij}\right)E\left[\mathbf{n}(k)^TM^{ij}\Pi M^{ij}\mathbf{n}(k)\right] \\ &= \sum_{i,j}\left(\frac{1}{m}p_{ij}\right)E\left[-\alpha^2\mathbf{n}(k)^TM^{ij}\mathbf{n}(k)\right] \\ &= \sum_{i,j}\left(\frac{1}{m}p_{ij}\right)(2\alpha^2\sigma^2)=2\alpha^2\sigma^2.\end{aligned} \tag{3.48}$$

We now have the following:

$$E\left[\mathbf{z}_L(k+1)^T\mathbf{z}_L(k+1)\right]=E\left[\mathbf{z}(k)^T\overline{\Omega}\mathbf{z}(k)\right]+2\alpha^2\sigma^2. \tag{3.49}$$

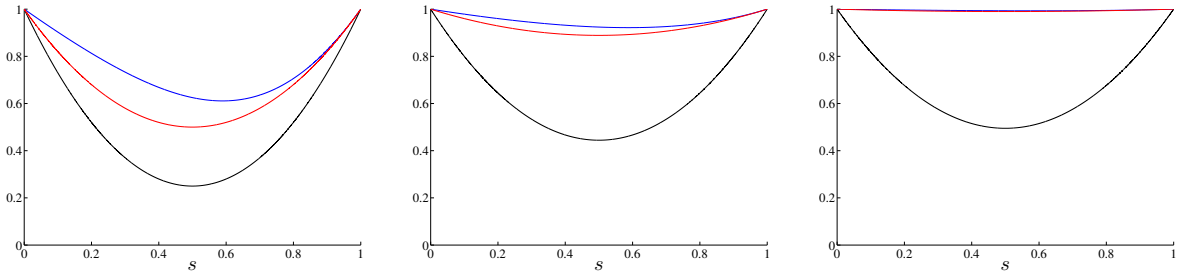
Let Ψ be the projection matrix onto $\mathcal{D}^\perp$, and define $\mathbf{z}_F:=\Psi\mathbf{z}$ as the follower-centric dispersion.[16] Then $\hat{L}=(\Pi+\Psi)\hat{L}$, and the second moment of $\mathbf{z}_L$ can be rewritten as:

$$E\left[\mathbf{z}_L(k+1)^T\mathbf{z}_L(k+1)\right]=E\left[\mathbf{z}_L(k)^T\overline{\Omega}\mathbf{z}_L(k)\right]+E\left[\mathbf{z}_F(k)^T\left(s^2\frac{\alpha^2}{m}\hat{L}\right)\mathbf{z}_F(k)\right]+2\alpha^2\sigma^2. \tag{3.50}$$

The second term on the right hand side shows the downfall of the model in its current form; any efficiency we try to create regarding leader-centric modes of dispersion is corrupted by the fact that the convergence of the second moment of $\mathbf{z}_L$ depends on the convergence of the second moment of $\mathbf{z}_F$. This term only disappears if we consider restricted leader-centric dispersion (or if $m=2$). We can explicitly illustrate how this dooms the general leader-centric dispersion convergence to be slow. First, we find the second moment of $\mathbf{z}_F$:

$$E\left[\mathbf{z}_F(k+1)^T\mathbf{z}_F(k+1)\right]=E\left[\mathbf{z}_F(k)^T\hat{\Omega}\mathbf{z}_F(k)\right]+E\left[\mathbf{z}_L(k)^T\left(s^2\frac{\beta^2}{m}\hat{L}\right)\mathbf{z}_L(k)\right]+2\beta^2\sigma^2, \tag{3.51a}$$

[14] These pairs correspond to the directed edges of the connectivity graph.
[15] We will soon show that this will be true for a symmetric star.
[16] Note that $\mathbf{z}_F=0$ for $m=2$.



(a) $m = 3$ (b) $m = 10$ (c) $m = 100$

Figure 3.2. Comparisons of eigenvalues determining convergence speed for three types of networks with $m = 3, 10$, and 100. Black: symmetric star with restricted leader-centric dispersion. Red: homogeneous complete graph with full dispersion. Blue: symmetric star with unrestricted leader-centric dispersion. Unrestricted leader-centric dispersion in the symmetric star has the slowest convergence speed, justifying the need for an updated model.

$$\hat{\Omega} = \Psi\left(I - 2\frac{s}{m}\hat{L}\right) + s^2\frac{\beta^2}{m}\hat{L}, \tag{3.51b}$$

$$\beta^2 = \sum_{k:\mathbf{v}_k \in \mathcal{D}^\perp} \left((\mathbf{e}_i - \mathbf{e}_j)^T \mathbf{v}_k\right)^2, \tag{3.51c}$$

where β^2 is only defined for i and j such that $p_{ij} \neq 0$ (as we did for α^2). For a symmetric star, it is easy to show that $\alpha^2 = m/(m-1)$ since $\mathbf{v}_A$ is the only eigenvector in $\mathcal{D}$. Additionally, it can be verified by induction on m that $\beta^2 = (m-2)/(m-1)$. The eigenvalues of $\overline{\Omega}$ and $\hat{\Omega}$ are given by:

$$\begin{aligned} \lambda_i\left(\overline{\Omega}\right) &= \begin{cases} 1 - \frac{s(2-s\alpha^2)}{m}\lambda_i(\hat{L}) & \text{if } \mathbf{v}_i \in \mathcal{D} \\ \frac{s^2\alpha^2}{m}\lambda_i(\hat{L}) & \text{if } \mathbf{v}_i \in \mathcal{D}^\perp \end{cases} \\ \lambda_i(\hat{\Omega}) &= \begin{cases} 1 - \frac{s(2-s\beta^2)}{m}\lambda_i(\hat{L}) & \text{if } \mathbf{v}_i \in \mathcal{D}^\perp \\ \frac{s^2\beta^2}{m}\lambda_i(\hat{L}) & \text{if } \mathbf{v}_i \in \mathcal{D} \end{cases} . \end{aligned} \tag{3.52}$$

Now we can write the following system of difference equations (for $m > 2$):

$$\mathbf{q}(k+1) \leq H\mathbf{q}(k) + J \tag{3.53a}$$

$$\mathbf{q}(k) = \left(E\left[\mathbf{z}_L(k)^T\mathbf{z}_L(k)\right] E\left[\mathbf{z}_F(k)^T\mathbf{z}_F(k)\right]\right)^T \tag{3.53b}$$

$$H = \begin{pmatrix} 1 - \frac{2m}{m-1}s + \left(\frac{m}{m-1}\right)^2 s^2 & \frac{m}{(m-1)^2}s^2 \\ \frac{m(m-2)}{(m-1)^2}s^2 & 1 - \frac{2}{m-1}s + \frac{m-2}{(m-1)^2}s^2 \end{pmatrix} \quad J = \frac{2\sigma^2}{m-1}\begin{pmatrix} m \\ m-2 \end{pmatrix}. \tag{3.53c}$$

For $s > 0$, there is coupling between the dynamics of the leader-centric and follower-centric subspaces of dispersion. The larger eigenvalue of H, $\lambda_1(H)$, determines the speed of convergence. In Figure 3.2, we compare this eigenvalue to two other eigenvalues. The first determines convergence for restricted leader centric dispersion in a symmetric star: $\lambda_L(\overline{W_r}) = 1 - 2s(1-s)\frac{m}{m-1}$. The second determines convergence for dispersion in a homogeneous complete graph: $\lambda_2(\overline{W_r}) = 1 - 4s(1-s)/(m-1)$.[17] We find that for all $m > 2$, $\lambda_1(H)$ is larger than both of these other eigenvalues, justifying our claim that the coupling of follower-centric and leader-centric moments

[17] Recall that this gives the fastest convergence speed for a regular gossip algorithm.

destroys the efficiency of the symmetric star graph when considering non-restricted leader-centric dispersion. At large m, $\lambda_L(\overline{W_r})$ for the symmetric star is much smaller than the two other eigenvalues, which motivates the need to find a way to take advantage of this efficiency even for unrestricted leader-centric dispersion.

3.2.4 Shortcomings of the Model

As noted above, the the coupling of dynamics for the second moments of leader-centric and follower-centric dispersions ruins convergence properties for leader-centric dispersion. This is an effect of a much more fundamental flaw of the model: the dynamics in their current form inherently create asymmetry in the system. More concretely, the flaw is that the leader only interacts with one of the followers at every iteration. Consider, for example, a symmetric star with two followers. Let the initial condition of the system be $\mathbf{x}(0) = (0, 1, -1)^T$, and let the first two iterations involve the first follower followed by the second follower. With $s = 1/2$, the state vector in the first two iterations will be $\mathbf{x}(1) = (1/2, 1/2, -1)^T$ and $\mathbf{x}(2) = (-1/4, 1/2, -1/4)^T$. Note that after interacting with both followers, the leader does not revert back to 0, the true mean of the system. This simple example is a more intuitive way to observe the fact that the follower-centric dynamics "pull" the leader away from convergence. We can resolve this issue by adding some synchronicity to the system. We present this partially-synchronous leader-centric gossip algorithm next.

3.3 A Partially Synchronous Leader-Centric Gossip Algorithm

The symmetric star has been shown to have promising potential as an efficient graph configuration for leader-centric dispersion. Now we update the algorithm of the leader-centric gossip algorithm over a symmetric star to eliminate the effects of slowly converging followers on the convergence of the leader. This new structure will motivate the design of hierarchical networks, networks that consist of hierarchies of symmetric star configurations.

We add synchronicity into the system by having the leader interact with the followers in "rounds." In each round, the leader interacts with each of the followers only once, although the order in which these interactions occur need not be fixed and can occur at random. Because nodes only exchange data with a single neighbor at any given time and the order of these interactions is random within each round, the algorithm maintains gossip-style characteristics. The "partial synchronicity" aspect has to do with the fact that although a leader coordinates its interactions with its followers within a round, multiple leaders at the same level of a hierarchy do not need to synchronize their rounds.

3.3.1 Model Dynamics

The most important element of this new algorithm is that for each round, the leader transfers data to the follower nodes with a fixed mean determined by the leader's mean at the beginning of the round. To motivate the new model consider again the example given in the last section: a symmetric star with two followers and initial condition $\mathbf{x}(0) = (0, 1, -1)^T$. At the end of the first round, we now have $\mathbf{x}(1) = (0, 1/2, -1/2)^T$, since the leader shares data with a mean of 0 to *both* followers. We see that this new model retains symmetry in the system; this concept manifests itself more rigorously in the fact that $\mathbf{z}_L$ now has the same dynamics as $\mathbf{z}$ and $\mathbf{x}$ (with a slightly different noise term). We start by writing the new convergence algorithm in the following way[18]:

$$\mathbf{x}(t) = Y\mathbf{x}(t-1) + \boldsymbol{\xi}(t), \tag{3.54a}$$

$$Y = I + s\sum_{j=2}^{m} M^{1j} = \overline{W_q} \text{ for a symmetric star, } q = (m-1)s, \tag{3.54b}$$

$$\boldsymbol{\xi}(t) = \sum_{j=2}^{m} M^{1j}\mathbf{n}_j(t) \tag{3.54c}$$

$$E\left[\mathbf{n}_j(t)\right] = 0, \;\; E\left[\mathbf{n}_i(t)\mathbf{n}_j^T(t)\right] = \sigma^2 I\delta(i-j). \tag{3.54d}$$

Note that t counts the rounds of the algorithm, and each round consists of the leader interacting with each of the $m-1$ followers. The total number of interactions is given by $k = (m-1)t$.

Convergence in Expectation

Because Y has the same eigenvectors as $\hat{L}$ for a symmetric star, convergence in expectation will hold as long as the eigenvalues have the correct structure: we need $\mathbf{1}$ to have an eigenvalue of 1 and all other eigenvalues to have magnitude < 1. The eigenvalues of Y are given by: 1 (with eigenvector $\mathbf{1}$), $1 - ms$ (with eigenvector $\mathbf{v}_A$), and (for $m > 2$) $1 - s$ (with eigenvectors $\mathbf{v}_{B_j}$). Therefore, 1 is a simple eigenvalue for $s > 0$, and it is the only eigenvalue with unit magnitude for $s < 2/m$. Then convergence in expectation holds for $s \in (0, 2/m)$.

[18]We will currently consider scalar data and will generalize to multidimensional data later on.

Convergence Speed for Leader-Centric Dispersion

Because Y acts like a scaled form of $\overline{W_s}$, many of the statements for the original gossip dynamics still hold: $\bar{x}$ is still invariant under the dynamics, $\mathbf{z} \perp \mathbf{1}$, and $\mathbf{z}$ follows the same dynamics as $\mathbf{x}$. However, now we can make a similar claim for $\mathbf{z}_L$:

$$\mathbf{z}_L(t) = \Pi\mathbf{z}(t) = \Pi Y\mathbf{z}(t-1) + \Pi\boldsymbol{\xi}(t) = Y\mathbf{z}_L(t-1) + \Pi\boldsymbol{\xi}(t), \tag{3.55}$$

where we have used the crucial fact that Y commutes with Π due to the symmetry of the dynamics. Now the second moment looks like the following:

$$\begin{aligned} E\left[\mathbf{z}_L(t+1)^T\mathbf{z}_L(t+1)\right] &= E\left[\mathbf{z}_L(t)^TY^2\mathbf{z}_L(t)\right] + E[\boldsymbol{\xi}(t+1)^T\Pi\boldsymbol{\xi}(t+1)] \\ &= (1-ms)^2E[\mathbf{z}_L(t)^T\mathbf{z}_L(t)] + E[\boldsymbol{\xi}(t+1)^T\Pi\boldsymbol{\xi}(t+1)]. \end{aligned} \tag{3.56}$$

The noise autocorrelation is given by:

$$\begin{aligned} E[\boldsymbol{\xi}(t)^T\Pi\boldsymbol{\xi}(t)] &= E\left[\left(\sum_{i=2}^m M^{1i}\mathbf{n}_i(t)\right)^T\Pi\left(\sum_{j=2}^m M^{1j}\mathbf{n}_j(t)\right)\right] \\ &= E\left[\sum_{j=2}^m \mathbf{n}_j^T(t)M^{1j}\Pi M^{1j}\mathbf{n}_j(t)\right] = \sum_{j=2}^m 2\alpha^2\sigma^2 = 2m\sigma^2. \end{aligned} \tag{3.57}$$

The noise autocorrelation is now bigger than that for the regular gossip algorithm (for $m > 2$) because effectively $m-1$ steps occur in each round. Importantly, this does not imply that the noise envelope is always larger for this new algorithm. Indeed, the growth of the noise envelope depends on the magnitude of the convergence eigenvalue, and $(1-ms)^2$ can be much smaller than $\lambda_2(\overline{W_r}) \geq 1 - 4s(1-s)/(m-1)$.

Now all the elements are in place to write the convergence time using Markov's inequality as we have done before. To apply Markov's inequality, we now have the following condition:

$$Q(t) := \mathbf{z}_L(t)^T\mathbf{z}_L(t) \geq 2m\sigma^2\frac{1-(1-ms)^{2t}}{1-(1-ms)^2} \tag{3.58}$$

Conditioning on the event Q yields the same results as above:

$$\begin{aligned} E\left[\mathbf{z}_L(t+1)^T\mathbf{z}_L(t+1)|Q(t+1)\right] &= (1-ms)^2E\left[\mathbf{z}_L^T(t)\mathbf{z}_L(t)|Q(t+1)\right] + 2m\sigma^2 \\ E\left[\mathbf{z}_L(t)^T\mathbf{z}_L(t)|Q(t)\right] &= (1-ms)^{2t}\mathbf{z}_L^T(0)\mathbf{z}_L(0) + 2m\sigma^2\frac{1-(1-ms)^{2t}}{1-(1-ms)^2} \end{aligned} \tag{3.59}$$

Finally, we can apply Markov's inequality conditioned on Q to get the convergence time:

$$k^* = (m-1)\left\lceil\frac{3\log\epsilon}{\log(1-ms)^2}\right\rceil. \tag{3.60}$$

The convergence time for the new algorithm has an extra factor of $m-1$ due to the previously mentioned reason that rounds consist of $m-1$ steps.[19]

[19]The expression for k^* is not defined for $s = 1/m$. We will cover this case in Section 3.3.5.

3.3.2 Data Selection

For this partially synchronous, leader-centric algorithm, data selection can still occur via random subset selection. The main difference now is that, for a given round, the leader must only choose data from the distribution that it had at the beginning of the round. In other words, it chooses $m-1$ random subsets without replacement from this original distribution. After the round is completed, it can mix the remaining data with the datasets it received from the $m-1$ followers. This method of data selection gives a practical reason to consider $s \in (0, 1/m]$ as opposed to $s \in (0, 2/m)$. We will give a more theoretical reason in Section 3.3.5.

3.3.3 Estimation of the Noise Envelope with an Updated Noise Model

As with the regular gossip algorithm, the assumption of stationary noise is inaccurate. Also, we have made another egregiously false assumption that the $m-1$ random samples from the leader's dataset in a given round are uncorrelated. Finally, due to the degree of centralization inherent in the new model dynamics, we now have the capability to consider separate noise values for different nodes. We will now address all of these issues. Note that, in particular, accounting for correlation of the leader's samples within a round can only *decrease* the size of the noise envelope, so it is advantageous to consider this more accurate method.

The main update we need to address is in the cross-correlation for $\mathbf{n}_i(t)$. The noise levels from different nodes within a round are still considered independent, as they are standard errors from different populations. However, now we account for cross-correlations between samples from the leader's dataset within a round, and we consider different noise levels for each node. Let $\sigma_i(t)$ denote the noise level for node i at round t. Then, we have for the noise cross-correlations:

$$E\left[\mathbf{n}_i(t)\mathbf{n}_i^T(t)\right] = \left(\sigma_1^2(t)\mathbf{e}_1\mathbf{e}_1^T + \sum_{j=2}^{m}\sigma_i^2(t)\mathbf{e}_j\mathbf{e}_j^T\right), \tag{3.61a}$$

$$E\left[\mathbf{n}_i(t)\mathbf{n}_j^T(t)\right] = c_{ij}(t)\mathbf{e}_1\mathbf{e}_1^T, \tag{3.61b}$$

where $c_{ij}(t)$ is the covariance between the means of different samples of the leader's data in round t. We can calculate this value efficiently using just one of the leader's samples, as we will show below.

With the new noise model, we now have an updated value for the noise autocorrelation:

$$\begin{aligned}
E[\boldsymbol{\xi}(t)^T\Pi\boldsymbol{\xi}(t)] &= E\left[\left(\sum_{i=2}^{m}M^{1i}\mathbf{n}_i(t)\right)^T\Pi\left(\sum_{j=2}^{m}M^{1j}\mathbf{n}_j(t)\right)\right] \\
&= E\left[\sum_{j=2}^{m}\mathbf{n}_j^T(t)M^{1j}\Pi M^{1j}\mathbf{n}_j(t)\right] + E\left[\sum_{i>1\neq j>1}\mathbf{n}_i^T(t)M^{1i}\Pi M^{1j}\mathbf{n}_j(t)\right] \\
&= \sum_{j=2}^{m}\alpha^2\left(\sigma_1^2(t)+\sigma_j^2(t)\right) + E\left[\sum_{i>1\neq j>1}\alpha^2\mathbf{n}_i^T(t)\left(\mathbf{e}_1-\mathbf{e}_i\right)\left(\mathbf{e}_1-\mathbf{e}_j\right)^T\mathbf{n}_j(t)\right] \\
&= \alpha^2\sum_{j=2}^{m}\left(\sigma_1^2(t)+\sigma_j^2(t)\right) + \alpha^2\sum_{i>1\neq j>1}c_{ij}(t).
\end{aligned} \tag{3.62}$$

Now, we show how to estimate all $\sigma_i^2(t)$ and $c_{ij}(t)$. Let $\omega_i^2(t)$ be the variance of the data in node i at the beginning of round t, and let $w_i^2(t)$ be the sample variance[20] of the random subset chosen by node i in round t. For the leader ($i = 1$), we can use any one of the $m - 1$ random subsets for round t to compute the sample variance. Then the true value of the noise $\sigma_i^2(t)$ as well as an estimate, $\hat{\sigma}_i^2(t)$, are simply given by the standard error formulae with a finite population correction factor:

$$\sigma_i^2(t) = \frac{\omega_i^2(t)}{sN}\left(1 - \frac{sN-1}{N-1}\right) = \frac{\omega_i^2(t)}{N-1}\frac{1-s}{s}. \tag{3.63a}$$

$$\hat{\sigma}_i^2(t) = \frac{w_i^2(t)}{sN}\left(1 - \frac{sN}{N}\right) = \frac{w_i^2(t)}{N}\frac{1-s}{s}. \tag{3.63b}$$

Now, $c_{ij}(t)$ is the covariance between the means of datasets that the leader shares with node i and j in round t. For the moment, we depart from our notational conventions and let $X_1^i, ..., X_{sN}^i$ and $X_1^j, ..., X_{sN}^j$ be the data points in these two samples. Let $x_1(t)$ be the true mean of the leader's data at round t, and let $\bar{X}^i$ and $\bar{X}^j$ be the sample means. Then, we have for $c_{ij}(t)$ as well as its estimate $\hat{c}_{ij}(t)$:

$$\begin{aligned} c_{ij}(t) = \mathrm{Cov}(\bar{X}^i(t), \bar{X}^j(t)) &= E\left[\bar{X}^i(t)\bar{X}^j(t)\right] - x_1^2(t) = \frac{1}{s^2N^2}E\left[\sum_{a,b} X_a^i(t)X_b^j(t)\right] - x_1^2(t) \\ &= E\left[X_1^i(t)X_1^j(t)\right] - x_1^2(t) = \mathrm{Cov}(X_1^i(t), X_1^j(t)) = -\frac{\omega_1^2(t)}{N-1}, \end{aligned} \tag{3.64a}$$

$$\hat{c}_{ij}(t) = -\frac{w_1^2(t)}{N}, \tag{3.64b}$$

where we have used the fact that all of the data points are identically distributed (but not independent), and so the covariance between any data point shared with node i and any data point shared with node j is the covariance of any two data points chosen from a simple random sample without replacement [21]. Also, observe that $c_{ij}(t) = -\sigma_1^2(t)\frac{s}{1-s}$ and $\hat{c}_{ij}(t) = -\hat{\sigma}_1^2(t)\frac{s}{1-s}$, so we can estimate any expressions that depend on $c_{ij}(t)$ and $\sigma_i(t)$ by simply replacing them by $\hat{c}_{ij}(t)$ and $\hat{\sigma}_i(t)$ respectively. We will exploit this fact to estimate the noise envelope.

Tracking the growth of the noise envelope requires keeping a running sum of noise terms with the discount factor of $(1 - ms)^2$ applied the appropriate number of times to each term. Importantly, this can de done in a decentralized way with each node keeping track of its own contribution to the envelope. First, note that the noise autocorrelation can be simplified to:

$$E[\boldsymbol{\xi}(t)^T\Pi\boldsymbol{\xi}(t)] = m\sigma_1^2(t)\left(1 - \frac{(m-2)s}{1-s}\right) + \frac{m}{m-1}\sum_{j=2}^{m}\sigma_j^2(t). \tag{3.65}$$

The correlation between the leader's datasets is taken into account in the term $\frac{(m-2)s}{1-s}$. For small s, the correlation has a negligible effect, but it can be as large as $\frac{m-2}{m-1}$ (which occurs at $s = \frac{1}{m}$).

[20] We adopt the convention that a sample of N values has only $N - 1$ degrees of freedom when computing the sample variance. Because of this convention, the sample variance is a biased estimator of the true variance when sampling without replacement: $E\left[w^2\right] = \frac{N}{N-1}\omega^2$. Hence, we will find that formulae for estimated values are different than their true values by a factor of $\frac{N-1}{N}$.

Denoting the noise envelope as K, the updated condition Q is now:

$$Q(t) := \mathbf{z}_L(t)^T \mathbf{z}_L(t) \geq K(t) \tag{3.66a}$$

$$K(t) := m\left(1 - \frac{(m-2)s}{1-s}\right)\sum_{a=1}^{t}\sigma_1^2(a)(1-ms)^{2(t-a)} + \frac{m}{m-1}\sum_{j=2}^{m}\sum_{a=1}^{t}\sigma_j^2(a)(1-ms)^{2(t-a)}. \tag{3.66b}$$

Recursive computation of the individual non-stationary noise envelopes (the components of K) is simple. At each time step t, each node multiplies its current envelope by $(1-ms)^2$ and then adds to this value its corresponding portion of $E[\boldsymbol{\xi}(t)^T \Pi \boldsymbol{\xi}(t)]$: either $\frac{m}{m-1}\sigma_j^2(t)$ for $j \neq 1$ or $m\sigma_1^2(t)\left(1 - \frac{(m-2)s}{1-s}\right)$ for the leader. Similarly, the dynamics of the second moment of leader-centric dispersion are given by:

$$\begin{aligned} E\left[\mathbf{z}_L(t+1)^T \mathbf{z}_L(t+1) | Q(t+1)\right] &= (1-ms)^2 E\left[\mathbf{z}_L^T(t)\mathbf{z}_L(t) | Q(t+1)\right] \\ &\quad + m\sigma_1^2(t+1)\left(1 - \frac{(m-2)s}{1-s}\right) + \frac{m}{m-1}\sum_{j=2}^{m}\sigma_j^2(t+1) \\ E\left[\mathbf{z}_L(t)^T \mathbf{z}_L(t) | Q(t)\right] &= (1-ms)^{2t}\mathbf{z}_L^T(0)\mathbf{z}_L(0) + K(t). \end{aligned} \tag{3.67}$$

This yields the same formula for k^* as before.

Estimation of the noise envelope no longer requires keeping track of the mean. Rather, each node needs to calculate $w_i^2(t)$ in each round and keep track of its component of the estimated noise envelope, $\hat{K}(t)$. This estimated noise envelope is the expression for $K(t)$ with $\sigma_i(t)$ replaced by $\hat{\sigma}_i(t)$. The leader simply needs to collect and add these values to get the aggregate (estimated) noise envelope. As outlined in the beginning of this subsection, this noise envelope now fully accounts for non-stationary noise, correlations between the leader's random samples within a round, and separate noise values for all nodes.

3.3.4 Multidimensional Data and Multiple Moments

The generalization to matching multiple moments and multidimensional data is essentially the same as before with a few notational differences. The dynamics are now given by:

$$\mathbf{x}(t) = (Y \otimes I_{dv})\,\mathbf{x}(t-1) + \boldsymbol{\xi}(t), \tag{3.68a}$$

$$\boldsymbol{\xi}(t) = \sum_{j=2}^{m}\left(M^{1j} \otimes I_{dv}\right)\mathbf{n}_j(t), \tag{3.68b}$$

$$E\left[\mathbf{n}_i(t)\right] = 0, \tag{3.68c}$$

$$E\left[\mathbf{n}_i(t)\mathbf{n}_i^T(t)\right] = \mathbf{e}_1\mathbf{e}_1^T \otimes \Sigma_1(t) + \sum_{j=2}^{m}\mathbf{e}_j\mathbf{e}_j^T \otimes \Sigma_i(t), \tag{3.68d}$$

$$\mathrm{diag}(\Sigma_i(t)) = \left(\sigma_{i,1u_1}^2(t), \sigma_{i,2u_1}^2(t), ..., \sigma_{i,du_1}^2(t), \sigma_{i,1u_2}^2(t), ..., \sigma_{i,du_v}^2(t)\right)^T, \tag{3.68e}$$

$$E\left[\mathbf{n}_i(t)\mathbf{n}_j^T(t)\right] = \mathbf{e}_1\mathbf{e}_1^T \otimes C_{ij}(t), \tag{3.68f}$$

$$\mathrm{diag}(C_{ij}(t)) = \left(c_{ij,1u_1}(t), c_{ij,2u_1}(t), ..., c_{ij,du_1}(t), c_{ij,1u_2}(t), ..., c_{ij,du_v}(t)\right)^T, \tag{3.68g}$$

where σ_{k,iu_j} is the noise level for the i-th dimension and the u_j-th moment in the k-th node. Similarly, c_{ab,iu_j} is the covariance between the means of the leader's samples during interaction with nodes a and b for the i-th dimension and the u_j-th moment.

Denote $\mathbf{z}_{L_{iu_j}}$ as the leader-centric dispersion in the i-th dimension and the u_j-th moment. Similarly, define $K_{iu_j}(t)$ as the corresponding noise envelope. The convergence time follows a similar pattern to the previous section, where we now look at the union of events over all d dimensions and v moments in each dimension. Denote the event R_{iu_j} as:

$$R_{iu_j}(t) := \frac{\|\mathbf{z}_{L_{iu_j}}(t)\|}{\|\mathbf{x}_{iu_j}(0)\|} \geq \hat{\epsilon}_{iu_j}(t) \tag{3.69a}$$

$$\hat{\epsilon}_{iu_j}(t) := \sqrt{\epsilon^2 + \frac{K_{iu_j}(t)}{\mathbf{x}_{iu_j}(0)^T \mathbf{x}_{iu_j}(0)}} \tag{3.69b}$$

and the corresponding Q_{iu_j} as:

$$Q_{iu_j}(t) := \mathbf{z}_{L_{iu_j}}(t)^T \mathbf{z}_{L_{iu_j}}(t) \geq K_{iu_j}(t). \tag{3.70}$$

Then the convergence criterion is given by $k^\dagger$ (which accounts for the $m-1$ steps in each round):

$$k \geq k^\dagger := (m-1)\left\lceil \frac{3\log\epsilon - \log d - \log v}{\log\left(1-ms\right)^2} \right\rceil \implies P\left(\bigcup_{j=1}^{v}\bigcup_{i=1}^{d} R_{iu_j}(t) \middle| \bigcap_{j=1}^{v}\bigcap_{i=1}^{d} Q_{iu_j}(t) \right) \leq \epsilon \tag{3.71}$$

3.3.5 Optimal s and Scalability

The expression for $k^\dagger$ is not valid for $s = 1/m$. This is because at this particular value of s, convergence occurs in one round ($m-1$ steps) regardless of the value for ϵ, d, and v. Other values of s near $1/m$ can give convergence in 1 round only if $k^\dagger = 1$, but this will depend on the choices for ϵ, d, and v. In this way, $s = 1/m$ is optimal even though it is not necessarily the unique optimum for all scenarios. We can see this optimality more clearly if we consider the expression for $k^\dagger$ without the ceiling operator. Then optimizing the value of s to minimize convergence time amounts to minimizing the value of $(1-ms)^2$ over the domain $s \in (0, 2/m)$. This is simply a parabola with a unique minimum of 0 at $1/m$. The symmetry of the convergence speed around $1/m$ means that we can effectively consider the domain $(0, 1/m]$ when optimizing s under budget constraints. We prefer sharing the least amount of data as possible.

Interestingly, unlike the regular gossip algorithm, this new partially synchronous leader-centric algorithm has a convergence speed that can scale with the number of followers, allowing the same number of rounds of convergence regardless of m. Indeed, for a given number of rounds desired for convergence, the amount of data that needs to be shared varies inversely with the number of followers. This scalability is a very favorable characteristic for a decentralized network algorithm.

3.4 Hierarchical Networks

So far, we have illustrated the dynamics for a single star configuration. Realistic networks cannot often be organized such that there is a single central leader and $m-1$ followers. As such, we seek to generalize the system presented thus far to more realistic scenarios while maintaining characteristics of the dynamics in this new algorithm. This objective naturally gives rise to the design of hierarchical networks, networks that consist of multiple levels of star configurations. We consider the hierarchical network to consist of $\mathcal{L}$ levels. Although it is not strictly necessary, for simplicity of analysis we will consider levels homogeneous: the number of followers for all leaders at a single level is uniform. With this level-wise homogeneity, we have at the top-most level ($l=1$) a star with m_1 nodes, and each of the m_1-1 followers in this level becomes a leader in $l=2$ with m_2-1 followers, etc.

This hierarchical system is both scalable and realistic. Scalability follows simply from the exponential growth in the number of layers: a seemingly modest four-level hierarchy with just $m_1=m_2=m_3=m_4=5$ nodes has $1+4+4^2+4^3+4^4=341$ total nodes, which is actually quite large for our purposes. More generally, the total number of nodes is given by:

$$m = 1 + \sum_{l=1}^{\mathcal{L}} \prod_{j=1}^{l} (m_j - 1). \tag{3.72}$$

This scalability advantage is likely what makes general hierarchical networks (i.e. not necessarily star or leader-centric configurations) common in many engineering applications and natural systems. Hierarchical routing is commonly used for Internet web traffic [12, 22], and the algorithms developed for this domain are directly applicable to leader selection in our network. Additionally, leader-centric hierarchies become particularly applicable when considering models of wireless networks, wherein a node is connected to all nodes within a certain physical radius [2, 6]. In such networks, hierarchical star configurations are easy to construct; we will consider this process more thoroughly along with tolerance of nodal failure when we discuss leader selection in Section 3.4.4.

3.4.1 Convergence to a Weighted Mean

The moment-matching process for a hierarchical network begins with the stars in level $\mathcal{L}$. The leaders in this tier indicate to their leaders (i.e. the leaders of level $\mathcal{L}-1$) when they have completed $k^\dagger$ steps so that the leaders of level $\mathcal{L}-1$ know when to begin their own moment-matching process. This continues up the hierarchy to $l=1$ (or just as far as necessary).[21] Although this design has obvious benefits of scalability and decentralization, it also modify convergence properties of the system. Namely, the moment-matching algorithm is agnostic to the level of the hierarchy in which it is being performed (hence its scalability). This means that when the star at $l=1$ performs moment matching, it treats all nodes equally and attempts to have the leader's mean converge to the mean of all nodes in the star. The major ramification of this feature is that after the entire moment-matching process is completed, the mean of the data in level 1's leader converges to a *weighted* mean of the entire dataset. Specifically, the means of nodes that reside higher in the hierarchy (i.e. at a lower level l) are weighted more heavily than those lower in the hierarchy. Let level i be the lowest level (i.e. largest level number) of the hierarchy in which a certain node

[21] In many networks, we may not need to perform moment matching for all $\mathcal{L}$ levels; there may be an inflection point in lifetimes, i.e. a level j such that nodes in levels $i \leq j$ are not prone to failure.

n participates.[22] Then, the weight assigned to n's original mean in the mean of the data for a leader at level $l \leq i$ is $\prod_{j=l}^{i} 1/m_j$.

The fact that convergence occurs to a weighted mean is not necessarily a negative aspect of the algorithm. The overall point of the moment-matching process is for unreliable nodes to share data with other nodes so that, in the event of nodal failure, the system is affected as little as possible. In converging to a weighted mean in the data at the level 1 leader, we have still accomplished the goal of propagating information across the network to the leader in a structured, provably convergent manner. The main difference now is in the behavior of node 1 when queried for a prediction on its dataset without communicating with other nodes (i.e. a local SoD prediction as opposed to the BCME prediction). With the regular gossip algorithm, the node would behave in a similar manner to a node which received a random subset of the entire dataset collected by the networked system. With this new hierarchical gossip algorithm, the node will behave differently, but not necessarily worse or better. We are willing to take this difference in performance due to the obvious benefits in overall communication cost and scalability compared to the regular gossip algorithm. In addition, the hierarchical tree-like setup optimizes communication time when performing training/predictions in the BCME.

3.4.2 Convergence Conditions for Hierarchical Networks

We now formally characterize the convergence of the moment-matching algorithm for a hierarchical system. As always, we start with scalar data and before generalizing to multidimensional data and multiple moments. We will consider the case that we want to match moments across all levels of the hierarchy so that the central leader of $l = 1$ has data that looks representative of the entire dataset across all nodes.[23] Additionally, for simplicity of analysis we consider homogeneous convergence criteria: there is a fixed ϵ for all levels in the hierarchy. As with level-wise homogeneity in the number of nodes, this constraint is not necessary, but it renders subsequent derivations more compact.

After running $k^\dagger$ steps of the original gossip algorithm, we were able to achieve convergence up to $\hat{\epsilon}$ with probability at least $1 - \epsilon$. Now, we look at the case when all symmetric star systems run $k^\dagger$ steps, so all systems reach $\hat{\epsilon}$ convergence with probability at least $(1 - \epsilon)^{m_L}$, where m_L is the number of leaders in the system:

$$m_L = 1 + \sum_{l=1}^{\mathcal{L}-1} \prod_{j=1}^{l} (m_j - 1). \tag{3.73}$$

Thus, we need a smaller ϵ in the hierarchical system to achieve the same level of confidence in convergence. Specifically, for a given ϵ_0 in the regular gossip algorithm, we now need

$$\epsilon = 1 - (1 - \epsilon_0)^{\frac{1}{m_L}}. \tag{3.74}$$

For small ϵ_0, we can write the first order approximation: $\epsilon \approx \epsilon_0/m_L$. This scaling approximately holds even for most ϵ_0 we will consider, as seen in Figure 3.3. We will use this approximation in further computations below.

[22] Note that the intermediate leaders belong to two levels of the hierarchy, whereas the leader of level 1 and the followers of level $\mathcal{L}$ belong to only one level.

[23] Depending on the reliabilities of nodes in the system, we may not need to perform moment matching across all levels, but rather only those levels with failure-prone nodes. The leader selection and hierarchy-building algorithm of Section 3.4.4 will tend to place more failure-prone nodes further from the hierarchy root (i.e. at greater l).

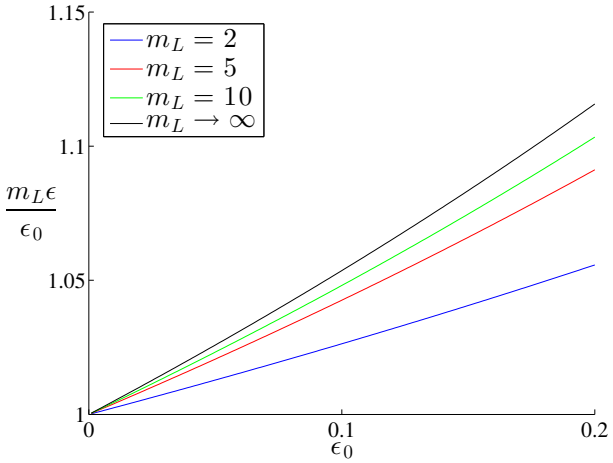


Figure 3.3. Scaling of ϵ with ϵ_0. Since $\frac{m_L\epsilon}{\epsilon_0} \approx 1$ for $\epsilon < 0.2$, we can justify the approximation $\epsilon \approx \epsilon_0/m_L$. As $m_L \to \infty$, $m_L\epsilon \to -\log(1-\epsilon_0)$.

Now, we calculate the level of convergence we are guaranteed for the whole system if all of the symmetric star systems converge. First consider noise-free dynamics. Denote the magnitude of dispersion for a leader at level l by $|z_1^l|$. This value is the magnitude of the distance between the leader's mean x_1^l and the mean of the the nodes in the star system associated with that leader. Suppose that we could bound $|z_1^l|$ in terms of $\|\mathbf{x}(0)\|$, the initial magnitude of the means for *all* nodes in the entire system. This would allow comparison of the error bound for the hierarchical system to that of the regular gossip algorithm. Indeed, suppose that we bound the error $|z_1^l|$ by $\delta_l(\epsilon)\|\mathbf{x}(0)\|$. We will solve for this function $\delta_l(\epsilon)$ shortly.

Importantly, $\delta_l(\epsilon)\|\mathbf{x}(0)\|$ bounds the error for $|z_1^l|$ from the mean value of its star system. Ideally, this would correlate with the weighted mean of all nodes under this leader in the entire system (i.e. the subtree underneath this leader). However, each of the intermediate leaders has its own error, and these errors propagate up the hierarchy. For this reason, define e_l as the worst case distance of x_1^l from the weighted mean of its subtree (denoted by $\bar{x}^l$):

$$\left|x_1^l - \bar{x}^l\right| < e_l. \tag{3.75}$$

We can easily write a recursive formula for e_l, since in the worst case the error $\delta_l(\epsilon)\|\mathbf{x}(0)\|$ of $|z_1^l|$ adds up with each of the e_{l+1}/m_l error terms from the follower nodes in the mean $\bar{x}^l$:

$$e_l = \delta_l(\epsilon)\|\mathbf{x}(0)\| + \frac{m_l - 1}{m_l} e_{l+1} \;\forall\, l < \mathcal{L} \tag{3.76a}$$

$$e_{\mathcal{L}} = \delta_{\mathcal{L}}(\epsilon)\|\mathbf{x}(0)\|. \tag{3.76b}$$

Now we find a value for $\delta_l(\epsilon)$.

Proposition 3.5. *Assume that all symmetric star systems within the hierarchy converge (which occurs with probability* $(1-\epsilon)^{m_L}$*). Define* $m_{max} := \max_i m_i$ *and* $\bar{\epsilon}_{max} := \epsilon\sqrt{\frac{m_{max}-1}{m_{max}}}$*. Then, a conservative value for* $\delta_l(\epsilon)$ *is* $\bar{\epsilon}_{max}(1+\bar{\epsilon}_{max})^{\mathcal{L}-l}$*.*

Proof. Due to the structure of $\mathbf{v}_A$, the leader-centric eigenvector, we know that $|z_1^l| = \sqrt{\frac{m_l-1}{m_l}}\|\mathbf{z}_L^l\|$.

Define $\bar{\epsilon}_l := \epsilon\sqrt{\frac{m_l - 1}{m_l}}$. For notational convenience, we will demonstrate the validity of the formula for $\delta_l(\epsilon)$ using a simple chain system, with just one leader and one follower at each level of the hierarchy. We will then show why this holds for general hierarchies.

With a chain hierarchy we can simplify notation. For $\mathcal{L}$ levels, we have $\mathcal{L}+1$ nodes. Let node $i \leq \mathcal{L}$ be the leader at level i. Let $x_i(k)$ and $z_i(k)$ be the mean and dispersion of node i at time k. Then, for the intermediate leaders (nodes 2 through $\mathcal{L}$), $x_i(k^\dagger)$ will be the mean of node i after having converged with the mean of node $i+1$, whereas $x_i(2k^\dagger)$ will be its mean after converging with node $i-1$. Recall that the gossip algorithm occurs first in level $\mathcal{L}$ and then moves up to $\mathcal{L}-1$, and so on. Consider the situation where we have performed convergence up to level $l < \mathcal{L}$ in the hierarchy. Then we can say the following:

$$|z_l(k^\dagger)| < \bar{\epsilon}_l \sqrt{x_l^2(0) + x_{l+1}^2(k^\dagger)}, \tag{3.77}$$

where the superscripts on x_i are exponents since our simplified notation does not use superscripts for state variables. For a general hierarchy, the radicand would include m_l terms. Our goal is to find a relation between the right hand side and $\|\mathbf{x}(0)\|$. Consider first the term $x_{l+1}^2(k^\dagger)$. If $l < \mathcal{L}-1$, then we have:

$$x_{l+1}^2(k^\dagger) \leq \left(\frac{x_{l+1}(0) + x_{l+2}(k^\dagger)}{m_{l+1}} \pm \bar{\epsilon}_{l+1} \sqrt{x_{l+1}^2(0) + x_{l+2}^2(k^\dagger)} \right)^2 := \left(a \pm \bar{\epsilon}_{l+1} b\right)^2. \tag{3.78}$$

Without loss of generality, we can consider $a \geq 0$ and consider just the larger term $(a + \bar{\epsilon}_{l+1}b)$.[24] Now we can apply Jensen's inequality to a uniform discrete random variable that takes each of the constituent values in b^2 with equal probability. Then, we get $a^2 \leq b^2/m_{l+1}$. Therefore,

$$x_{l+1}^2(k^\dagger) \leq b^2 \left(\frac{1}{\sqrt{m_{l+1}}} + \bar{\epsilon}_{l+1} \right)^2. \tag{3.79}$$

This process can occur again, as we now have $x_{l+2}^2(k^\dagger)$ as a term in b^2 (for general hierarchies there would be $m_{l+1} - 1$ such terms). It will continue until we reach an expression for $x_{\mathcal{L}}^2(k^\dagger)$. Equivalently, if we had $l = \mathcal{L} - 1$, we would have had just the following:

$$x_{\mathcal{L}}^2(k^\dagger) \leq \left(x_{\mathcal{L}}^2(0) + x_{\mathcal{L}+1}^2(0)\right) \left(\frac{1}{\sqrt{m_{\mathcal{L}}}} + \bar{\epsilon}_{\mathcal{L}} \right)^2. \tag{3.80}$$

In this way, we see that for $l < \mathcal{L}$, we have:

$$|z_l(k^\dagger)| < \bar{\epsilon}_l \sqrt{x_l^2(0) + \sum_{i=l+1}^{\mathcal{L}} \left(x_i^2(0) \prod_{j=l+1}^{i} \left(\frac{1}{\sqrt{m_j}} + \bar{\epsilon}_j \right)^2 \right) + x_{\mathcal{L}+1}^2(0) \prod_{j=l+1}^{\mathcal{L}} \left(\frac{1}{\sqrt{m_j}} + \bar{\epsilon}_j \right)^2} \tag{3.81}$$

Even though we have performed this computation with just two nodes per level, we have been careful to keep the computation general, retaining the use of m_l instead of 2. In this way, our computation applies to any arbitrary hierarchy. In the above expression, instead of just one $x_i^2(0)$ term in the summation, we would have $m_{i-1} - 1$ such terms. Also, we would have $m_{\mathcal{L}} - 1$ terms similar to $x_{\mathcal{L}+1}^2(0)$. We can get a very conservative upper bound by applying the amplification

[24] For $a < 0$, we can multiply by -1 and relabel terms.

$\prod_{j=l+1}^{\mathcal{L}}(1+\bar{\epsilon}_j)^2$ to every node in the system (which is an upper bound on the actual maximum amplification to any individual term):

$$|z_l(k^\dagger)| < \bar{\epsilon}_l\|\mathbf{x}(0)\| \prod_{j=l+1}^{\mathcal{L}}(1+\bar{\epsilon}_j) < \bar{\epsilon}_{max}\|\mathbf{x}(0)\|\,(1+\bar{\epsilon}_{max})^{\mathcal{L}-l} \tag{3.82}$$

Finally, we consider $l = \mathcal{L}$, which is simple:

$$|z_{\mathcal{L}}(k^\dagger)| < \bar{\epsilon}_{\mathcal{L}}\|\mathbf{x}(0)\| < \bar{\epsilon}_{max}\|\mathbf{x}(0)\|, \tag{3.83}$$

so we see that the formula applies to all $l \leq \mathcal{L}$. □

Now we can use the expression for $\delta_l(\epsilon)$ to get an upper bound for the total error e_l. Define $c := \frac{m_{max}-1}{m_{max}}$. Then we can unroll the recursive expression in Equation 3.76:

$$\begin{aligned} e_l &< \bar{\epsilon}_{max}\|\mathbf{x}(0)\|(1+\bar{\epsilon}_{max})^{\mathcal{L}-l}\sum_{i=0}^{\mathcal{L}-l}\left(\frac{c}{1+\bar{\epsilon}_{max}}\right)^i \\ &< \bar{\epsilon}_{max}\|\mathbf{x}(0)\|(1+\bar{\epsilon}_{max})^{\mathcal{L}-l}\sum_{i=0}^{\mathcal{L}-l}\left(\frac{1}{1+\bar{\epsilon}_{max}}\right)^i \\ &= \|\mathbf{x}(0)\|\left((1+\bar{\epsilon}_{max})^{\mathcal{L}-l+1}-1\right). \end{aligned} \tag{3.84}$$

The maximum error will be at the leader, which is

$$e_1 < \|\mathbf{x}(0)\|\left((1+\bar{\epsilon}_{max})^{\mathcal{L}}-1\right) \approx \mathcal{L}\bar{\epsilon}_{max}\|\mathbf{x}(0)\| \approx \sqrt{\frac{m_{max}-1}{m_{max}}}\epsilon_0\frac{\mathcal{L}}{m_L}\|\mathbf{x}(0)\| \leq \epsilon_0\|\mathbf{x}(0)\|. \tag{3.85}$$

Thus, we see that even the very conservative upper bounds we have used have resulted in the total error e_1 being at most $\frac{\mathcal{L}}{m_L}\sqrt{\frac{m_{max}-1}{m_{max}}}$ of the error bound we would have in the regular gossip algorithm. This fraction is approaches 1 (for a large single-level system), but it is often much smaller. The intuitive reason for the much tighter convergence of the hierarchical system is the fact that the normalizing constant for convergence in each star subunit includes only those members of that star subunit. Since each star subunit performs convergence with respect to a smaller normalization factor, the entire system performs convergence with better accuracy compared to the regular gossip algorithm.

The generalization to including noise as well as matching multiple moments and dimensions is straightforward. As before, the extra moments and dimensions all converge as systems in parallel, so we have a separate convergence window e_{1,iu_j} for the i-th dimension and the u_j-th moment. We can bound all of these windows by redefining $\bar{\epsilon}_{max}$ as

$$\bar{\epsilon}_{max} := \sqrt{\frac{m_{max}-1}{m_{max}}}\max_k\max_i\max_{u_j}\hat{\epsilon}^k_{iu_j}, \tag{3.86}$$

where $k \in \{1,2,...,m_L\}$ and $\hat{\epsilon}^k$ is the error window for the k-th star subunit within the hierarchy (including the noise envelope) computed at the end of the moment-matching process. These (scalar) values can be propagated up the hierarchy, as the leader of each star subunit can determine its own noise envelope by collecting noise and magnitude values from its followers (as per Section

3.3.3), and it can then compare this envelope to those from the lower levels of its own subtree. Thus, the leader at level 1 can can easily determine $\bar{\epsilon}_{max}$, and the final convergence windows are bounded by

$$e_{1,iu_j} < \|\mathbf{x}_{iu_j}(0)\| \left((1+\bar{\epsilon}_{max})^{\mathcal{L}} - 1\right). \tag{3.87}$$

With the inclusion of the noise envelope, comparison to the regular gossip algorithm is no longer as simple as in the noise-free case, as it depends on all of the parameters that define the noise envelope. Sample calculations on modest to moderate size systems (10-100 nodes and 2-5 levels in the hierarchy) have resulted in the hierarchical system having a convergence window either on the same order of magnitude or smaller than that of the regular gossip algorithm. We also reiterate that the convergence bound we have presented is extremely conservative and represents a worst-case scenario of all errors adding up from multiple levels of the hierarchy.

3.4.3 Absolute Speed Comparisons with the Regular Gossip Algorithm

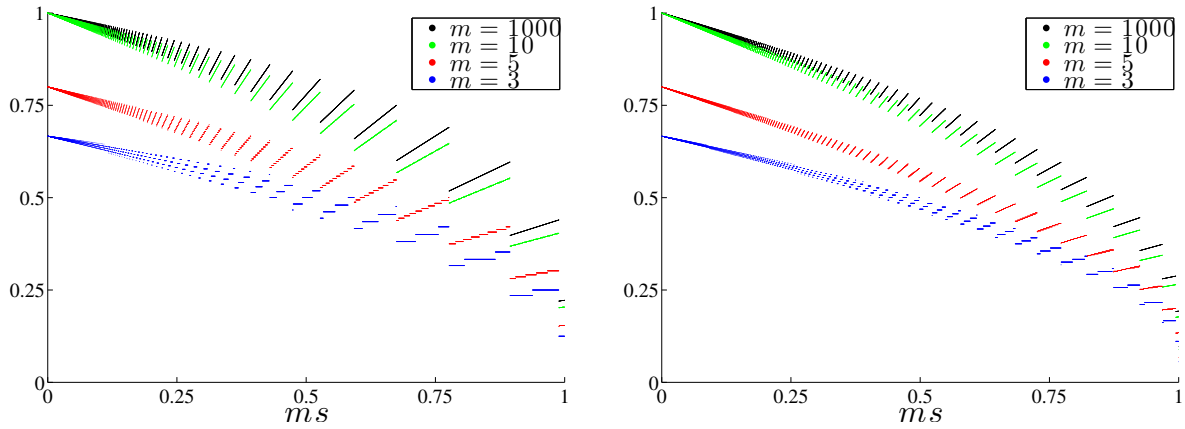
Now we compare the absolute speed of convergence for the regular gossip algorithm with the hierarchical, leader-centric algorithm. Note that we compare *absolute* speeds. For the regular gossip algorithm, multiple iterations of the algorithm can occur in parallel, the exact number of which depends on the graph topology. For the hierarchical system, all moment-matching processes for a single level occur in parallel, but the interactions between a single leader and its followers occur in series. In this way, the absolute speed is a measure of how much clock time (measured in units of iterations) is required for convergence.

Complete Graphs The complete graph is the graph topology with the fastest convergence speed for the regular gossip algorithm. It is only relevant to compare this system with a single-level hierarchical system since a star system is a subgraph of a complete graph. In other words, if a network can behave as a complete graph, it can certainly behave as a star by simply ignoring many edges. The best case scenario for a regular gossip algorithm over a complete graph is that $\lfloor m/2 \rfloor$ iterations occur simultaneously, so the absolute time is the number of required iterations divided by this factor. Denote this as T_r, and denote T_h as the absolute convergence time for the hierarchical system. With just one level, $T_h = k^{\dagger}$. The expression for T_h/T_r is given by:

$$\frac{T_h}{T_r} = \frac{(m-1)\left\lfloor \frac{m}{2} \right\rfloor \left\lceil \frac{3\log f}{2\log(1-ms)} \right\rceil}{\left\lceil \frac{3\log f}{\log(1-4s(1-s)/(m-1))} \right\rceil}, \quad f := \frac{\epsilon_0}{(dv)^{\frac{1}{3}}}. \tag{3.88}$$

Figure 3.4 shows T_h/T_r for various m. We observe that the hierarchical system convergence time is faster than that of the regular gossip algorithm for all m shown, but both algorithms converge at times that are roughly the same order of magnitude for $ms < 0.95$. Importantly, this better convergence speed comes at a fraction of the communication cost, since the star system has $1/m$ as many edges as the complete graph.

Structured Hierarchy of Complete Graphs Now we compare a structured hierarchy consisting of complete graphs that are sparsely connected. We create this graph in the following manner: we begin with a hierarchical system with $\mathcal{L} = 3$ and $m_1 = m_2 = m_3 := a$ nodes for each level. Then, we let all nodes within each star subunit be completely connected. In other words, we let all followers of a specific star subunit be connected not only to their respective leader but also to all other followers within that star subunit. This results in a hierarchy of complete graphs. For the regular gossip algorithm, we optimize $\lambda_2(\overline{W_r})$ (or equivalently $\lambda_2(\hat{L})$) through the convex



(a) $\epsilon_0/(dv)^{\frac{1}{3}} = 0.05$ (b) $\epsilon_0/(dv)^{\frac{1}{3}} = 0.001$

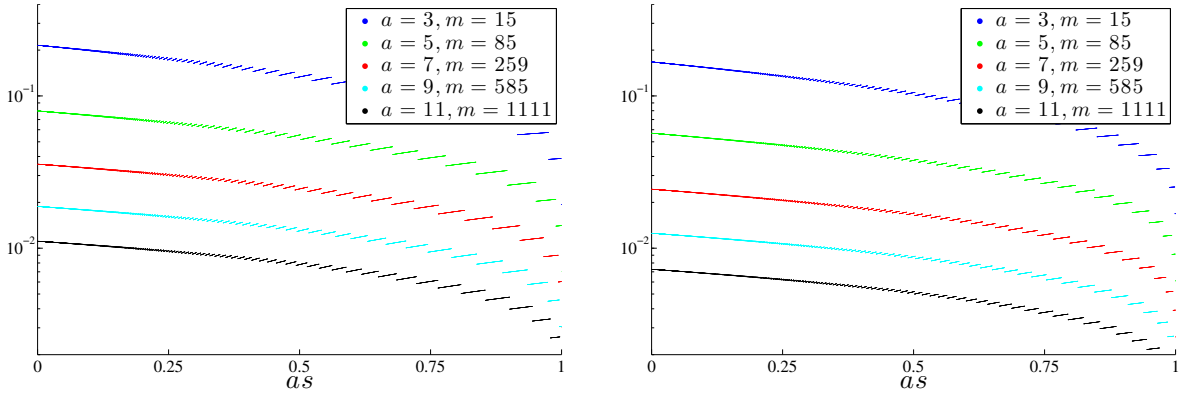
Figure 3.4. T_h/T_r vs. ms for various m for the complete graph. Convergence time depends on ϵ_0, d, and v only through the ratio $\epsilon_0/(dv)^{\frac{1}{3}}$. Decreasing this ratio decreases the "step" length of the trends (as seen in the comparison from (a) to (b).). These jagged "steps" occur due to the ceiling operators in the expressions for T_h and T_r. The star system converges faster than the regular gossip algorithm over the complete graph at a fraction of the communication cost.

optimization package CVX [4, 5]. For $a > 2$, an optimal graph structure is the following: the leader at $l = 1$ looks at all followers with equal probability, all followers at $l = 1$ look at each other with equal probability, and the rest of the system is structured exactly as in the star hierarchy. This structure implies that the maximum number of simultaneous iterations is if all leaders at $l = 3$ are active with one of their followers, and all leaders at $l = 2$ either exchange with each other or node 1. This results in a maximum number of simultaneous iterations of $(a-1)^2 + \lfloor a/2 \rfloor$. For the hierarchical system, the absolute convergence time is equivalent to running $\mathcal{L} = 3$ star subsystems, each with a nodes, in series. The expression for T_h/T_r is now:

$$\frac{T_h}{T_r} = \frac{3(a-1)\left((a-1)^2 + \left\lfloor \frac{a}{2} \right\rfloor\right)\left\lceil \frac{3\log(f/m_L)}{2\log(1-as)} \right\rceil}{\left\lceil \frac{3\log f}{\log\left(1-2\lambda_2(\tilde{L})s(1-s)/m\right)} \right\rceil}, \quad f := \frac{\epsilon_0}{(dv)^{\frac{1}{3}}}. \tag{3.89}$$

This ratio is shown in Figure 3.5 for various a. The star system converges one to three orders of magnitude faster than the regular gossip algorithm for $as < 0.95$. This dramatic performance difference is due to the fact that the regular gossip algorithm is constrained in speed by the sparse links connecting the complete subunits. This clearly illustrates the overwhelming advantage of dealing only with the eigenvalues associated with $\mathcal{D}$ in the leader-centric, hierarchical gossip algorithm.

Ad-Hoc Wireless Networks Finally, we compare the performance of the two gossip algorithms over geometric random graphs, introduced by [6], in which nodes are connected to all other nodes within a certain physical radius. Such graphs are often used to model ad-hoc wireless networks. To analyze this type of network, we draw m points uniformly at random on a unit square and choose a radius R as the threshold of connectivity. For the regular gossip algorithm, we again optimize $\lambda_2(\overline{W_r})$ using CVX. As opposed to using an exhaustive search for the maximum number of simultaneous iterations, we do so in a greedy fashion: we randomly remove pairs of neighbors until it is no longer possible to do so, repeat this procedure multiple times, and then denote the maximum value as b.



(a) $\epsilon_0/(dv)^{\frac{1}{3}} = 0.05$ (b) $\epsilon_0/(dv)^{\frac{1}{3}} = 0.001$

Figure 3.5. T_h/T_r vs. as for various a for the structured hierarchy of complete graphs. Decreasing the ratio $\epsilon_0/(dv)^{\frac{1}{3}}$ decreases the "step" length of the trends (as seen in the comparison from (a) to (b)) and also decreases the ratio T_h/T_r slightly. The leader-centric, hierarchical gossip algorithm converges 1 to 3 orders of magnitude faster than the regular gossip algorithm for $as < 0.95$.

For the hierarchical gossip algorithm, we assume that each edge in the graph has unit weight and that the leader has been predetermined as the node with the largest closeness.[25] The subgraph for communication used by the hierarchical system is the single-source shortest path tree centered at the leader. We are no longer guaranteed level-wise homogeneity, so we denote m_{i_j} as the number of nodes in the j-th star subsystem of level i. Let T_{h_i} be the overall convergence time for the i-th level of the hierarchy (i.e. the overall time for all parallel star subsystems in level i to converge):

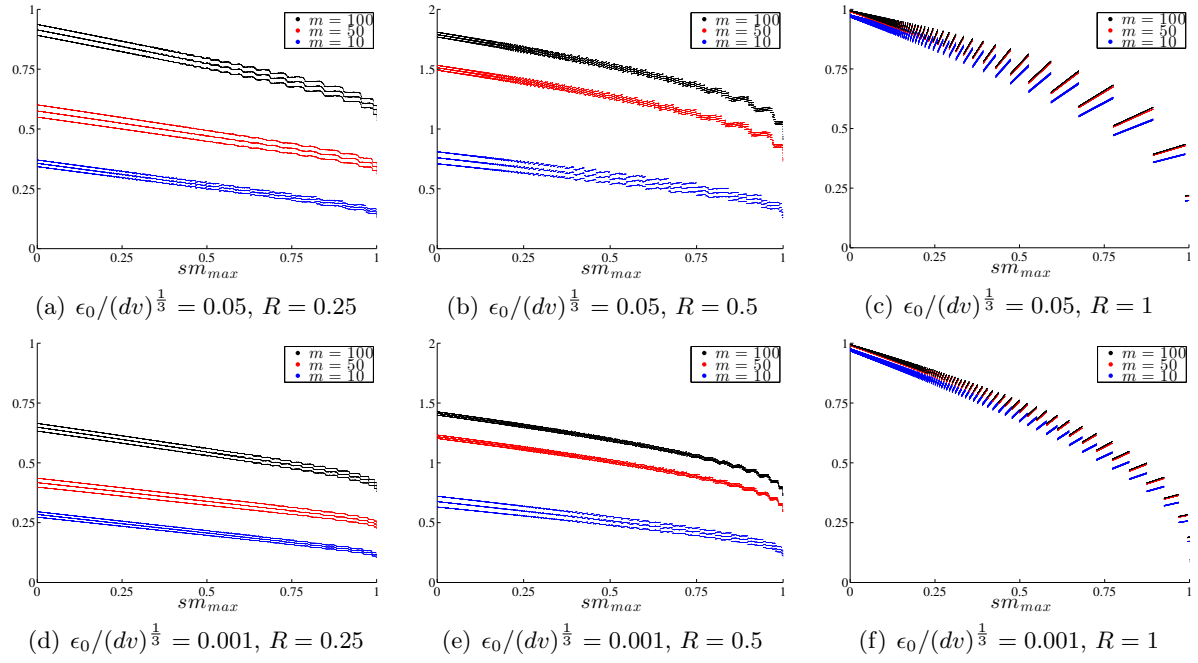
$$T_{h_i} = \max_j \left((m_{i_j} - 1) \left\lceil \frac{3\log(f/m_L)}{2\log\left(1 - m_{i_j} s\right)} \right\rceil \right), \qquad f := \frac{\epsilon_0}{(dv)^{\frac{1}{3}}}, \tag{3.90}$$

where m_L is still defined as the total number of leaders in the system but is no longer given by Equation 3.73 due to the level-wise heterogeneity. We assume a fixed amount of data transfer for every iteration, i.e. a constant s for all levels in the hierarchy and for the regular gossip algorithm. This implies $s \in (0, 1/m_{max}]$, where $m_{max} := \max_i \max_j m_{i_j}$. Then the ratio T_h/T_r is given by:

$$\frac{T_h}{T_r} = \frac{b \sum_{i=1}^{\mathcal{L}} T_{h_i}}{\left\lceil \frac{3\log f}{\log\left(1 - 2\lambda_2(\tilde{L}) s(1-s)/m\right)} \right\rceil}. \tag{3.91}$$

Figure 3.6 shows results for various m and R. For each pair of m and R, we average results over 25 random graphs drawn using the process denoted above. The hierarchical system gains the largest speed advantage in the sparsest graphs (i.e. small R), as we saw in the structured hierarchy of complete graphs (Figure 3.5). The nearly-complete limit (very large R) results in trends that are equivalent to those of Figure 3.4. The transition between these two limits (intermediate R values) can result in cases where $T_h > T_r$, particularly when m is large. At these intermediate R, the graph is connected enough that the overhead of incrementally converging $\mathcal{L}$ levels of the hierarchy can be large enough to render the regular gossip algorithm faster. This effect is magnified for larger m because increasing m for a given intermediate value of R increases $\mathcal{L}$ (on average).

[25] The closeness of a node is the inverse of the sum of its shortest path distances to all other nodes. We discuss this metric in Section 3.4.4.



(a) $\epsilon_0/(dv)^{\frac{1}{3}} = 0.05$, $R = 0.25$ (b) $\epsilon_0/(dv)^{\frac{1}{3}} = 0.05$, $R = 0.5$ (c) $\epsilon_0/(dv)^{\frac{1}{3}} = 0.05$, $R = 1$

(d) $\epsilon_0/(dv)^{\frac{1}{3}} = 0.001$, $R = 0.25$ (e) $\epsilon_0/(dv)^{\frac{1}{3}} = 0.001$, $R = 0.5$ (f) $\epsilon_0/(dv)^{\frac{1}{3}} = 0.001$, $R = 1$

Figure 3.6. T_h/T_r vs. sm_{max} for ad-hoc wireless networks (geometric random graphs). Three values of m are shown for each of three values of R and two values of $\epsilon_0/(dv)^{\frac{1}{3}}$. Results from 25 trials are shown with the mean value $\pm$ the standard error.
As before, decreasing the ratio $\epsilon_0/(dv)^{\frac{1}{3}}$ decreases the "step" length of the trends (as seen in the vertical comparisons (a)-(d), (b)-(e), and (c)-(f)) and also decreases the ratio T_h/T_r slightly. The large R limit results in mostly complete graphs and the small R limit results in very sparse graphs; $T_h < T_r$ in both scenarios as seen in Figures 3.4 and 3.5. Intermediate values of R can result in $T_h > T_r$, when the overhead of converging $\mathcal{L}$ levels in series is larger than that associated with performing the regular gossip algorithm.

3.4.4 Leader Selection

In comparing the speed of convergence for the regular gossip algorithm with that of the hierarchical, leader-centric gossip algorithm over various network topologies, we assumed that for the latter algorithm, the hierarchical structure was already known. In some applications this may certainly be the case, as many realistic systems are designed hierarchically. However, in many other applications we may need the system to autonomously elect a leader and a resulting hierarchical structure. We now discuss this problem of leader selection.

A good leader must have both centrality and reliability. Centrality characterizes the notion that the leader can communicate efficiently with all other nodes, whereas reliability implies the leader will live long enough to collect and combine predictions from other nodes when queried with a test dataset.[26] Simultaneously satisfying both of these characteristics poses an interesting design problem for which we develop the following approach: we choose a leader by finding the most central node in a graph that encodes node reliabilities. We define what we mean by centrality and the graph of reliabilities below.

[26]For hierarchical systems, centrality will often correlate with minimizing the depth of the hierarchy, which also has obvious benefits in terms of convergence speed.

Closeness Centrality in the Reliability Graph

We define a cost function $c(S) > 0$ that determines the cost c_i of communicating with node i based on its expected lifespan S_i. This cost can be thought of as an inverse to reliability. Essentially, this metric needs to capture the fact that nodes with smaller expected lifetimes are less reliable and have a higher associated cost. Therefore, this function can be as simple as $c(S) = 1/S$, or it can be a monotonic transformation of $1/S$. One particularly useful form could be to pass $1/S$ through a sigmoid, which would set saturation levels for maximum and minimum reliabilities and would softly separate nodes into two classes (reliable and unreliable). We then form the "reliability graph" by assigning the weight c_i to all neighborly connections directed from node i to node j. For each node i, we then solve the single-source shortest paths problem: we determine the smallest distance $d(i, j)$ from node i to node j over directed edges. The elected leader is then the node with the smallest total sum of distances, p_i. Specifically, the leader (which is node 1 by our previous conventions) solves the following problem:

$$p_1 = \sum_{k \neq 1} d(1, k) \leq \sum_{j \neq i} d(i, j) = p_i \; \forall \; i \in \{1, 2, ..., m\}. \tag{3.92}$$

The resulting (non-unique) optimal paths that form $d(1, j)$ are then used to construct the hierarchy. In the case of ties for the leader, we can define a protocol that systematically decides how to break the tie; one component could be to minimize hierarchy depth. Note that the sum p_i is always greater than or equal to the length of the shortest path tree rooted at i, and it implicitly penalizes hierarchy depth by counting the link from i to j every time j lies along the shortest path to another node k. This measure of centrality, p_i, is coined the farness of a node in network theory; its inverse is the closeness of a node [1, 16].

Decentralized and Efficient Computation of Shortest Path Trees

The main challenge of the leader selection and hierarchy-building process above is the decentralized and efficient computation of shortest paths from a source to all other nodes. This is the same problem tackled by adaptive routing protocols used to dynamically route traffic in computer networks (such as the Internet).[27] Popular protocols are mostly variations and sophistications of the distributed Bellman-Ford algorithm (i.e. distance-vector protocols) or Dijkstra's algorithm (i.e. link-state protocols) for determining shortest path trees. Specifically, many real-world implementations are particularly adapted to handle node failures, changing network topologies, and naturally hierarchical systems.[28] If a leader at some level l fails, the network can reroute and determine a new leader relatively quickly. In contrast, if a node fails during moment matching with the regular gossip algorithm, we need to recompute $\lambda_2(\overline{W_r})$ for the new network topology, a lengthy and involved process.

Such nodal failure is an important reason why we force nodes to always have N data points as opposed to simply giving data to the leader: if the leader fails during moment matching, a new node must assume its position and continue the process, so we want to avoid losing more data

[27] Another similarity to our approach is the cost function. For routing protocols, the cost is usually chosen to be inversely proportional to the bandwidth of a link to a node.

[28] Examples of distance-vector protocols include the Routing Information Protocol (RIP) (`http://tools.ietf.org/html/rfc2453`) and Babel (`http://tools.ietf.org/html/rfc6126`). Examples of link-state protocols are the Open Shortest Path First (OSPF) protocol (`http://tools.ietf.org/html/rfc2328`) and the Optimized Link State Routing (OLSR) protocol (`http://tools.ietf.org/html/rfc3626`). A comprehensive overview of these algorithms can be found in [12, 22].

than necessary. Even if the leader of the whole system fails, we do not necessarily need to redo moment matching for all $\mathcal{L}$ levels. We only need to redo the process for the star subsystems that change leadership. If the global leader fails during BCME training/prediction, we need to redo the global step of the training process.

3.4.5 Communication Time Complexity of the Moment-Matching Process

We can easily calculate upper bounds for the communication time involved in the moment-matching process as measured by the number of data points transferred. Within a star subsystem with a nodes, the total amount of data transfer is $2sNk^\dagger$, which we can bound by the following:

$$2sNk^\dagger = 2sN(a-1)\left\lceil \frac{3\log(f/m_L)}{2\log\left(1-as\right)}\right\rceil < 2Nx\left\lceil \frac{3\log(f/m_L)}{2\log\left(1-x\right)}\right\rceil, \tag{3.93a}$$

$$x := as, \qquad f := \frac{\epsilon_0}{(dv)^{\frac{1}{3}}}. \tag{3.93b}$$

The continuous function $-x/\log(1-x) < 1 \;\forall\; x \in [0,1]$, but the ceiling operator adds a slight complication to our scenario. We have found empirically that $x\lceil -C/\log(1-x)\rceil < C \;\;\forall\, x \in [0,1]$ when $C > 2$. This constraint is equivalent to requiring $\log(m_L/f) > 4/3$. This only adds a binding constraint on f for small m_L. Specifically, for $m_L = 1$, 2, and 3, f is constrained to be < 0.27, 0.53, and 0.80, respectively. Since $f < 1$ by design, larger m_L do not enforce binding constraints. With these constraints in place, we can write the bound without x:

$$2sNk^\dagger < 3N\log(m_L/f). \tag{3.94}$$

All star subsystems of a given level i in the hierarchy perform moment matching in parallel, so the total data transfer time is bounded by:

$$\text{Data Transfer Time} < 3N\mathcal{L}\log(m_L/f) = 3\frac{N^G}{m}\mathcal{L}\log(m_L/f), \tag{3.95}$$

where $N^G = Nm$ is the total number of data points. In the worst case of a chain hierarchy, $\mathcal{L} = m_L = m-1 < m$; in the best case of a single star, $\mathcal{L} = m_L = 1$, so we can find corresponding upper bounds for each case purely in terms of m, N^G, d, v, and ϵ_0:

$$\text{Data Transfer Time (Worst Case)} < 3N^G(\log m + \log f^{-1}), \tag{3.96a}$$

$$\text{Data Transfer Time (Best Case)} < 3\frac{N^G}{m}\log f^{-1}. \tag{3.96b}$$

The "average" case scenario of a tree with level-wise homogeneity and $m_i = b > 2 \;\forall\; i$ has a convenient form in the limit of $b \gg 1$ and $\mathcal{L} \gg 1$.[29] In this limit, $m \approx b^{\mathcal{L}}$ and $m_L \approx m/b$, whereby:[30]

$$\text{Data Transfer Time (Average Case)} < 3\frac{N^G}{\log b}\frac{\log m}{m}(\log m - \log b + \log f^{-1}). \tag{3.96c}$$

[29] Recall that $m_i = 2$ corresponds to a chain network.

[30] The actual formulae for m and m_L are (for $b > 2$):

$$m = \frac{(b-1)^{\mathcal{L}+1}-1}{b-2}, \qquad m_L = \frac{(b-1)^{\mathcal{L}}-1}{b-2}.$$

Importantly, the data transfer time is always upper bounded by an expression that is linear in N^G, logarithmic in d, and at most logarithmic in m. This implies that the moment-matching process is scalable to large, high-dimensional datasets and large networks.

3.5 Discussion

We have developed a scalable approach to improving the robustness of a network to nodal failure. The fundamental idea of the method is simple and intuitive: exchanging data points until all nodes have "similar" subsets of data. The novelty of our algorithm is in providing guarantees for performance and for developing the formalism to prescribe how long the data exchanging process should occur. Finally, we have improved upon the existing gossip algorithm of [2] to suit our leader-centric needs, resulting in significant boosts in performance. We consider techniques to improve our method and make it more applicable to realistic networks in Chapter 4 below.

Chapter 4

Conclusions and Further Work

This study focused on developing distributed protocols to perform efficient GP regression in networked systems. Although scalability and decentralization were important aspects of our research efforts, they were not the overall goals. Rather, our work tackled two main themes associated with distributed GP regression: accuracy and robustness. Accuracy was defined in terms of the $SMSE$ and $SNLP$ error metrics through comparisons to centralized approaches to GP regression; robustness was defined in terms of the ability of the system to cope with nodal failure (and a subsequent loss of data). We designed two distinct protocols to address each theme, and we enforced scalability and decentralization in each design. The BCME method performs accurate GP regression, and the moment-matching protocol improves robustness to nodal failure.

The BCME model combines the flexibility of mixture-of-experts models with the scalability of sparse GP methods. Specifically, we sophisticated the BCM to account for separate hyperparameters in each node and developed a novel approach to choosing inducing inputs. Experimental evaluations indicate that the BCME performs competitively with popular sparse GP methods when we consider accuracy with respect to cost (as measured by hyperparameter training time). Indeed, although the BCME does not necessarily perform as well as centralized sparse methods for a given M, it can afford to greatly increase M without significantly impacting cost. The experiments also made manifest the limits and tradeoffs associated with the BCME. Namely, our method loses numerical stability at large M and m. Furthermore, there are inherent tradeoffs between accuracy and cost as well as accuracy and robustness.

An important characteristic of the BCME is its ability to easily incorporate improvements. We can diminish the magnitude of the accuracy/cost tradeoff by simply making smaller networks (small m) more computationally attractive. We have already alluded to an approach to accomplish this task: make each node a sparse GP (such as FITC or Var). This technique results in each node having local or "private" inducing inputs as well as the global inducing inputs required by the BCME formalism. Whether these private and global variables should be equivalent or distinct is an interesting question requiring further research. Furthermore, we can improve the numerical stability problem and diminish the accuracy/robustness tradeoff by making larger networks (large m) more numerically stable. A simple technique for this is to create a truly hierarchical BCME. The current BCME acts as if the network structure is a star (even though it most likely is not), since all nodes share inducing inputs with a single global leader. Instead, we can let separate star subsystems act as mini-BCME models, and we can combine these in a level-wise manner similar to our approach for moment matching. Then, insofar as the numerical computations are

concerned, the effective size of the network is never larger than $\max_j \max_i m_{i_j}$, the maximum size of a star subsystem.

The moment-matching protocol increases the robustness of the system to nodal failure by prescribing a structured technique for nodes to share data with each other. The algorithm extends an existing asynchronous gossip algorithm to form a partially synchronous, leader-centric variant that is more suited to the demands of the BCME regression network. Our formalism takes into account the noise associated with trading random subsets of data. In addition, we handle "effectively" multivariate data, which includes cases wherein data is truly multivariate, cases wherein we want to match multiple moments of scalar data, and all variants in between. Finally, we have developed an approach to leader selection that balances reliability with centrality. Theoretical results and simulations illustrate clear speed advantages over the original gossip algorithm, and the overall time complexity of the moment-matching process is at most linear in all relevant variables associated with a system's data and network size.

Like the BCME, the moment-matching protocol is amenable to various improvements. Although the process is already efficient, we can further boost performance by first learning about the connection between inputs and outputs. In its most practical connotation, robustness to nodal failure implies that we want to resist losing data that matters for regression. In this sense, we should first learn about which input dimensions impact prediction, which will yield a more accurate value for the effective dimensionality of the data while matching moments. One technique to perform this inference is for each node to quickly and coarsely fit hyperparameters on a very small random subset of its data. Additionally, we can ameliorate the accuracy/robustness tradeoff by accounting for heterogeneous N^i in our analysis of convergence time. No longer requiring nodes to have the same amount of data in our formalism will make our protocol more compatible with clustering. A very simple way to accomplish this task is the following: each star subsystem determines which node has the smallest number of data points, and then all other nodes within that subsystem choose only a random subset of that many data points to perform moment-matching. In this approach, all nodes still appear to have the same amount of data for the moment-matching protocol, but nodes with particularly large clusters can keep the rest of their data aside for learning a separate set of hyperparameters.

Having outlined numerous frontiers to explore in future research, we pause to reflect on the significance of our findings. Our two protocols are very distinct and even disparate in their respective approaches. Nevertheless (or even perhaps because of this characteristic), they can easily work in tandem with each other in many realistic network scenarios. Indeed, depending on the specific application, we can perform moment matching for as many levels and for as small an ϵ as necessary before performing regression with the BCME. Additionally, although we have focused on nodal failures, our methods are flexible enough to handle other kinds of changes in network topology, such as node additions and leadership exchanges. Overall, we believe that our distributed protocols have the ability to develop into a mature framework for scalable, distributed nonparametric regression. In this way, the implications of expanding upon our methods reach across a range of fields and applications.

Appendix A

Computationally Efficient Form for FITC/PITC

The Sherman-Morrison-Woodbury formula for matrix inversion is given by:

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U\left(C^{-1}+VA^{-1}U\right)^{-1}VA^{-1}, \tag{A.1}$$

where A and C are invertible but not necessarily of the same size, and U and V are appropriately sized rectangular matrices. This expression is particularly applicable to the inversion of $\Lambda + Q_{ff}$. Define $S := \left(K_{uu} + K_{uf}\Lambda^{-1}K_{fu}\right)^{-1}$. Using the Sherman-Morrison-Woodbury formula and exploiting the substitution $K_{uf}\Lambda^{-1}K_{fu} = S^{-1} - K_{uu}$, we can rewrite the FITC/PITC predictive distribution as:

$$\begin{aligned} P(\mathbf{f}_*|\mathbf{y}, X, X_u, X_*) &= \mathcal{N}\left(Q_{*f}\left(Q_{ff}+\Lambda\right)^{-1}\mathbf{y}, K_{**} - Q_{*f}\left(Q_{ff}+\Lambda\right)^{-1}Q_{f*}\right) \\ &= \mathcal{N}\left(K_{*u}SK_{uf}\Lambda^{-1}\mathbf{y}, K_{**} - Q_{**} + K_{*u}SK_{u*}\right). \end{aligned} \tag{A.2}$$

For FITC, inversion of Λ is $O(N)$ since it is diagonal, so training time is dominated by the computation of $K_{uf}\Lambda^{-1}K_{fu}$ inside S, which takes $O(NM^2)$ time. For PITC, the blocks in Λ can be as large as $M \times M$, such that the inversion of all N/M blocks takes $O(N/M \times M^3) = O(NM^2)$ time if the inversions are performed in serial. If the inversion of each block is computed in parallel (which is possible in the BCM formulation), the blocks can be as large as $b \times b$, where $b = \lfloor (NM^2)^{1/3} \rfloor$. If the blocks have size $c \times c$, with $c > b$, then training time is $O(c^3)$ if the inversions are done in parallel and $O(Nc^2)$ if they are done in serial. Prediction time for the mean is dominated by the computation of K_{*u}, which is $O(M)$ per test case. Once this is computed, prediction of the variance is dominated by calculation of Q_{**} and $K_{*u}SK_{u*}$, which are both $O(M^2)$ if S is precomputed. In practice, numerical stability can be improved by using Cholesky decompositions (where appropriate) instead of inverting matrices explicitly.

Appendix B

Equivalence between PITC and BCM

Observing the equivalence between the BCM and PITC is not trivial, even though it merely involves algebraic manipulations. We show below how to transform the expression for the PITC predictive distribution into the form presented in the BCM. It is easiest to do so using the computationally efficient expression developed in Appendix A.

To begin, we compute the predictive distribution on the inducing inputs. This is equivalent to letting $\mathbf{f}_* = \mathbf{u}$:

$$\begin{aligned} P(\mathbf{u}|\mathbf{y}, X, X_u) &= \mathcal{N}\left(Q_{uf}\left(Q_{ff}+\Lambda\right)^{-1}\mathbf{y}, K_{uu} - Q_{uf}\left(Q_{ff}+\Lambda\right)^{-1}Q_{fu}\right) \\ &= \mathcal{N}\left(K_{uu}SK_{uf}\Lambda^{-1}\mathbf{y}, K_{uu} - Q_{uu} + K_{uu}SK_{uu}\right) \\ &= \mathcal{N}\left(K_{uu}SK_{uf}\Lambda^{-1}\mathbf{y}, K_{uu}SK_{uu}\right) \end{aligned} \tag{B.1}$$

As we did in Equation 2.18, we will denote this posterior distribution on the inducing inputs as $\mathcal{N}\left(\mu^G(\mathbf{u}), \Sigma^G(\mathbf{u})\right)$. Recall from Appendix A that $S^{-1} := K_{uu} + K_{uf}\Lambda^{-1}K_{fu}$. Due to the block diagonal structure of Λ, we can rewrite this expression. Let Λ^i correspond to the i-th block of Λ (so $\Lambda^i = K_{f^if^i} - Q_{f^if^i} + \sigma_n^2 I$ is *not* sparse). Then, we have:

$$\begin{aligned} S^{-1} &= K_{uu} + \sum_{i=1}^{m} K_{uf^i}(\Lambda^i)^{-1}K_{f^iu} \\ &= -(m-1)K_{uu} + \sum_{i=1}^{m}\left(K_{uu} + K_{uf^i}(\Lambda^i)^{-1}K_{f^iu}\right) \\ &= -(m-1)K_{uu} + (S^i)^{-1}, \end{aligned} \tag{B.2}$$

where we have defined S^i as a local version of S that depends only on $\mathbf{f}^i$. Having established this relation, the posterior precision matrix can now easily be rewritten:

$$\Sigma^G(\mathbf{u})^{-1} = K_{uu}^{-1}S^{-1}K_{uu}^{-1} = -(m-1)K_{uu}^{-1} + \sum_{i=1}^{m}\left(K_{uu}S^iK_{uu}\right)^{-1}. \tag{B.3}$$

Finally, using the Sherman-Morrison-Woodbury formula, we recognize that

$$\begin{aligned} K_{uu}S^iK_{uu} &= K_{uu} - K_{uf^i}\left(Q_{f^if^i} + \Lambda^i\right)^{-1} Kf^iu \\ &= K_{uu} - K_{uf^i}(K_{f^if^i} + \sigma_n^2 I)^{-1}K_{f^iu} = \Sigma^i(\mathbf{u}). \end{aligned} \tag{B.4}$$

Thus, we recover Equation 2.18b for $\Sigma^G(\mathbf{u})$. Similarly, we can rewrite the posterior mean as:

$$\begin{aligned} K_{uu}SK_{uf}\Lambda^{-1}\mathbf{y} &= \Sigma^G(\mathbf{u})K_{uu}^{-1}\sum_{i=1}^m \left(K_{uf^i}(\Lambda^i)^{-1}\mathbf{y}^i\right) \\ &= \Sigma^G(\mathbf{u})\sum_{i=1}^m \left(K_{uu}^{-1}(S^i)^{-1}K_{uu}^{-1}K_{uu}S^iK_{uf^i}(\Lambda^i)^{-1}\mathbf{y}^i\right) \\ &= \Sigma^G(\mathbf{u})\sum_{i=1}^m \left(\Sigma^i(\mathbf{u})^{-1}K_{uu}S^iK_{uf^i}(\Lambda^i)^{-1}\mathbf{y}^i\right). \end{aligned} \tag{B.5}$$

As above, we use the Sherman-Morrison-Woodbury formula to recognize that

$$\begin{aligned} K_{uu}S^iK_{uf^i}(\Lambda^i)^{-1}\mathbf{y}^i &= K_{uf^i}\left(Q_{f^if^i} + \Lambda^i\right)^{-1}\mathbf{y}^i \\ &= K_{uf^i}\left(K_{f^if^i} + \sigma_n^2 I\right)^{-1}\mathbf{y}^i = \mu^i(\mathbf{u}), \end{aligned} \tag{B.6}$$

whereby we can recover Equation 2.18c for $\mu^G(\mathbf{u})$.

The predictive distribution on test points is now simple, as we see that the mean is given by:

$$\mu^G(\mathbf{f}_*) = K_{*u}SK_{uf}\Lambda^{-1}\mathbf{y} = K_{*u}K_{uu}^{-1}K_{uu}SK_{uf}\Lambda^{-1}\mathbf{y} = K_{*u}K_{uu}^{-1}\mu^G(\mathbf{u}), \tag{B.7}$$

and the covariance is given by:

$$\begin{aligned} \Sigma^G(\mathbf{f}_*) &= K_{**} - Q_{**} + K_{*u}SK_{u*} = K_{**} - Q_{**} + K_{*u}K_{uu}^{-1}K_{uu}SK_{uu}K_{uu}^{-1}K_{u*} \\ &= K_{**} - Q_{**} + K_{*u}K_{uu}^{-1}\Sigma^G(\mathbf{u})K_{uu}^{-1}K_{*u}, \end{aligned} \tag{B.8}$$

which are the BCM expressions of Equation 2.19.

Appendix C

Evaluating Various Aspects of the BCME Model

C.1 Random X_u vs. Optimized X_u

We evaluate the change in accuracy with respect to optimization of X_u. We do so by first fixing the values of $\boldsymbol{\theta}$ at a separately optimized value, initializing X_u as a random subset of data, and then optimizing X_u. We repeat this experiment 5 times and evaluate the error measures before and after optimization of X_u. Specifically, we look at the percent reduction in error, denoted as D_{SMSE} and D_{SNLP}:

$$D_{SMSE} = 100\frac{SMSE_R - SMSE_O}{SMSE_R}, \tag{C.1a}$$

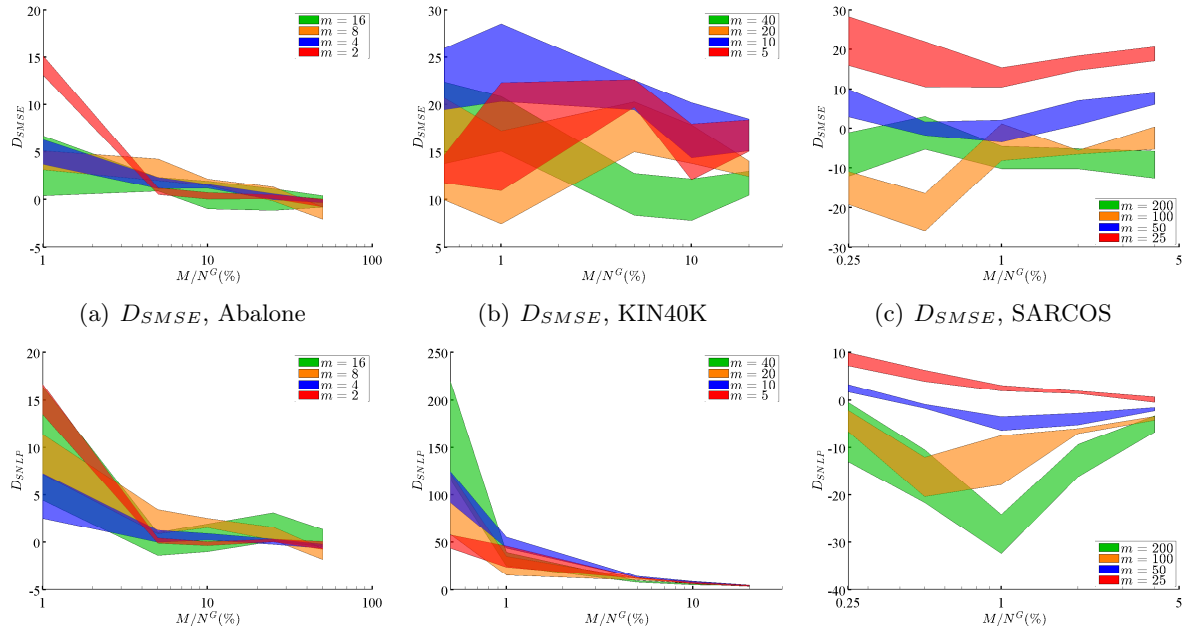
$$D_{SNLP} = 100\frac{SNLP_R - SNLP_O}{|SNLP_R|}, \tag{C.1b}$$

where $SMSE_R$ and $SNLP_R$ are error measures with random X_u, and $SMSE_O$ and $SNLP_O$ are error measures with optimized X_u. The results are shown in Figure C.1 for clustered data and Figure C.2 for randomly distributed data.

In general, we find that the effects of optimizing X_u decrease with increasing M. This is a result of the fact that as M increases, the optimization procedure has less work to accomplish: in the limit of $M/N^G = 1$, the optimization stops after one iteration, as the inducing set perfectly describes the data according to the loss function has a value of 0 (the smallest achievable value). Nevertheless, the magnitudes of the effects for a given M depend on various aspects of the datasets and the data partitioning, as we now explain.

For Abalone, the effects of optimizing X_u are relatively small: all changes for either the SMSE or SNLP with clustered or randomly distributed data are within $\pm 15\%$. We have seen that Abalone is essentially a linear dataset, so it is difficult to make drastic improvements in errors.

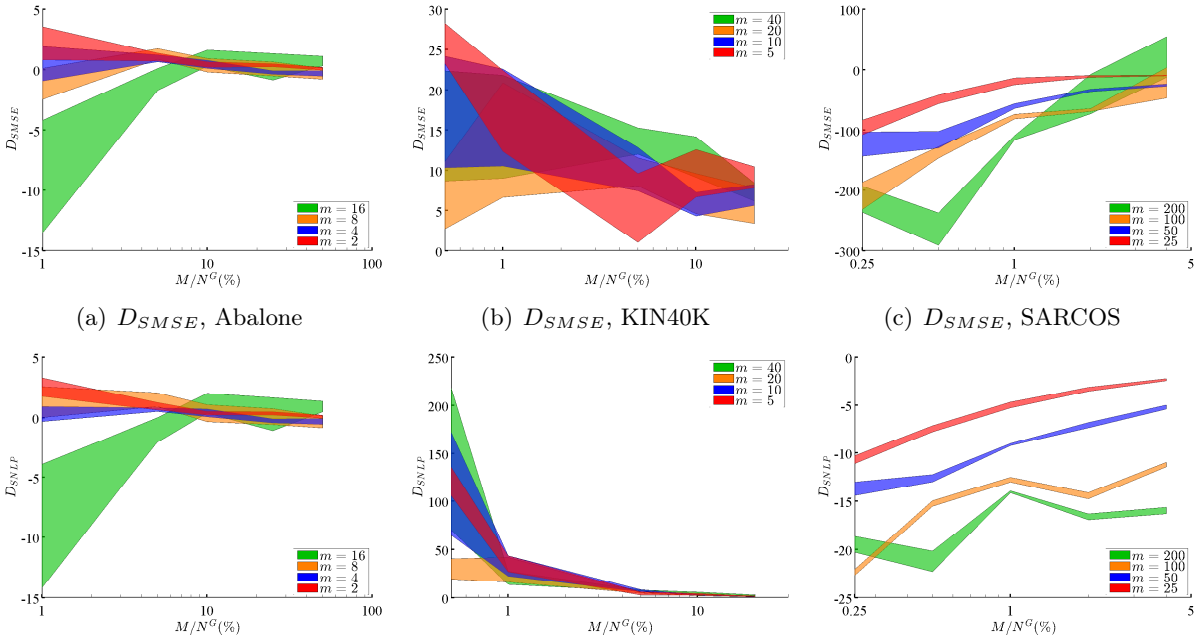
The effects of optimizing X_u are a bit more interesting for KIN40K and SARCOS due to the nonlinearity of the datasets. For the most part, the optimization of X_u has positive or negligibly negative effects on the errors. The overwhelming exception is Figure C.2(c), which illustrates a crucial characteristic of the optimization procedure. Optimization of X_u results in up to $100\% - 200\%$ increases in $SMSE$. It turns out this is a coupling of a few different issues. First, we recall that the optimization of X_u attempts to find a compressed version of the original input



(a) D_{SMSE}, Abalone (b) D_{SMSE}, KIN40K (c) D_{SMSE}, SARCOS

(d) D_{SNLP}, Abalone (e) D_{SNLP}, KIN40K (f) D_{SNLP}, SARCOS

Figure C.1. Error measures with random X_u vs. optimized X_u for clustered data. Colored regions represent the mean value ± the standard error over 5 trials.

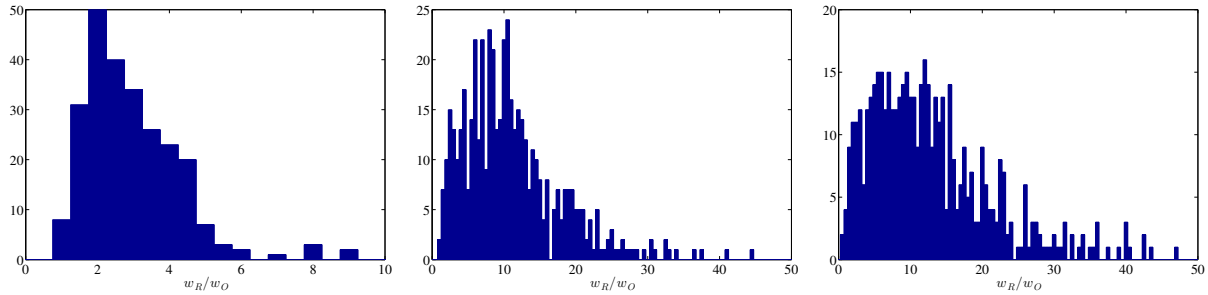


(a) D_{SMSE}, Abalone (b) D_{SMSE}, KIN40K (c) D_{SMSE}, SARCOS

(d) D_{SNLP}, Abalone (e) D_{SNLP}, KIN40K (f) D_{SNLP}, SARCOS

Figure C.2. Error measures with random X_u vs. optimized X_u for randomly distributed data. Colored regions represent the mean value ± the standard error over 5 trials.

data distribution. When each node has a random subset of data, their optimized X_u locations will tend to fall into similar places, thereby reducing the effective value of M. So why do we not see this problem with KIN40K? First, the problem of picking similar X_u is worse in the case



(a) KIN40K,$m = 5, \frac{M}{N^G} = 0.5\%$ (b) SARCOS,$m = 25, \frac{M}{N^G} = 0.25\%$ (c) SARCOS,$m = 25, \frac{M}{N^G} = 0.25\%$ Projected into relavant subspace

Figure C.3. Histograms of w_R/w_O, the ratio of distances between the closest X_u locations of different nodes before and after optimization. Bin widths are fixed at 0.5. For (c), we project points into the subspace relevant to the outputs before calculating distances. The optimized X_u locations get much closer together on average in SARCOS than in KIN40K (i.e higher w_R/w_O), especially in the subspace relevant to the outputs. This explains the significant increase in $SMSE$ with optimized X_u observed in Figure C.2(c).

of SARCOS due to the input data distributions. KIN40K has a roughly uniform distribution, whereas SARCOS is roughly normal in all dimensions. Thus, for SARCOS, more inducing points will tend to lie close to the mean of the distribution. Second, we have found that only 8 of the 21 input dimensions in SARCOS have a contribution to the output, as determined by examination of the hyperparameters trained on a subset of data. Therefore, the optimization routine wastes time moving points in directions orthogonal to this relevant subspace and can therefore unwittingly make the points very close in this subspace despite being far apart in $\mathbb{R}^d$. This brings the effective value of M even lower, as the points get even closer in the subspace that matters to the outputs. We verify this reasoning in Figure C.3. For every inducing point within node i, let w be the Euclidean distance to closest inducing point from any other node $j \neq i$. In this way, we define w_R as the value before optimizing X_u and w_O as the value after optimization. We plot w_R/w_O to show how much closer the X_u locations have become to each other, an indicator of the effective decrease in M. Even though KIN40K also moves inducing points closer together (i.e. $w_R/w_O > 1$), the effects are not strong enough to make the same appreciable degradation in performance that we see with SARCOS.

The upshot of this analysis is simple. Optimizing X_u has positive or acceptably negative effects on errors with clustered data, but it can have very negative effects on errors with randomly distributed data. Therefore we should perform optimization of X_u locations if the data is clustered to any extent. If the data is randomly distributed, we should choose X_u randomly and not optimize the locations. In future work, we may be able to leverage dimensionality reduction before or in parallel with the X_u optimization procedure, similar to the work of [19].

C.2 Separate vs. Joint Optimization of X_u and $\boldsymbol{\theta}$ for Standard GP Nodes

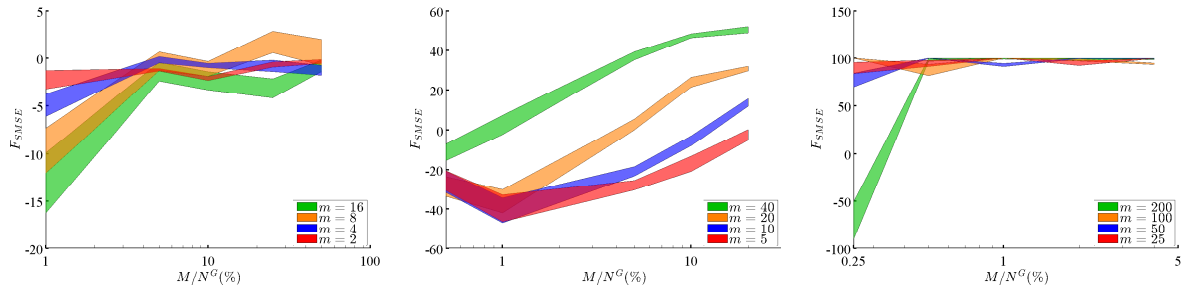
In Section C.1, we separately optimized $\boldsymbol{\theta}$ to test only the effects of optimizing X_u. Now we study the regularization effects of X_u on predictive performance when individual nodes are standard GP models. As noted earlier, we expect the regularization (i.e. joint optimization of X_u^i and $\boldsymbol{\theta}^i$) to have negative effects on unclustered data, but it is difficult to predict the effects on clustered data. Regularization may prevent overfitting by local experts, but when the inducing parameter set is too small, this may result in oversmoothing. We test every experimental setup 5 times with separate optimization and 5 times with joint optimization. We choose to illustrate the percent difference in error with respect to joint optimization:

$$F_{SMSE} = 100\frac{SMSE_J - SMSE_S}{SMSE_J}, \tag{C.2a}$$

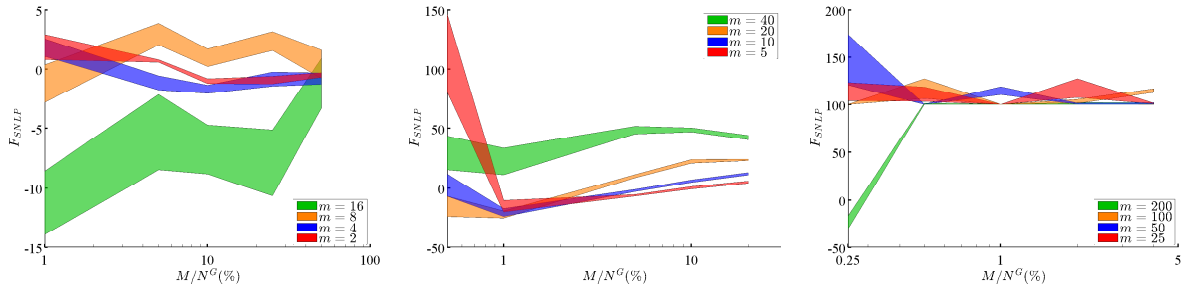
$$F_{SNLP} = 100\frac{SNLP_J - SNLP_S}{|SNLP_J|}, \tag{C.2b}$$

where $SMSE_J$ and $SNLP_J$ are errors with joint optimization, and $SMSE_S$ and $SNLP_S$ are errors with separate optimization. Unlike Equation C.1, which compared single trials of random/optimized X_u and took the resulting mean and standard error, we now compare the means of all 5 trials. For example, $SMSE_J$ is given by $x \pm \delta x$, where x is the mean $SMSE$ of all 5 trials using joint optimization, and δx is the standard error. In this way, uncertainties in F_{SMSE} and F_{SNLP} are computed by propagating errors in quadrature. Figure C.4 shows the results for clustered data, and Figure C.5 shows the results for randomly distributed data. We have included the latter for completeness, even though the results of the previous section indicate that we should not bother optimizing X_u at all when data is randomly distributed.

Again, we see that changes in errors for Abalone are negligible due to its linearity. The effects for KIN40K and SARCOS are as expected for unclustered data: the regularization effects over-smooth the model with joint optimization, and separate optimization is able to improve error measures drastically. We also see similar benefits to separately optimizing X_u and $\boldsymbol{\theta}$ for clustered data. The only situations in which separate optimization performs appreciably worse than joint optimization is the limit of large m and small M (most notably for SARCOS). As noted in Section 2.3.2, with large m and clustered data, each node has a small amount of data covering a small subregion of input space. The nodes can therefore be prone to overfitting hyperparameters in which case regularization is beneficial. The overfitting effect is assuaged by larger M, as we observe. Overall, the data indicate separate optimization has positive effects away from the extreme limit of large m and small M. We should separately optimize X_u and $\boldsymbol{\theta}$ regardless of the degree of clustering of the data.

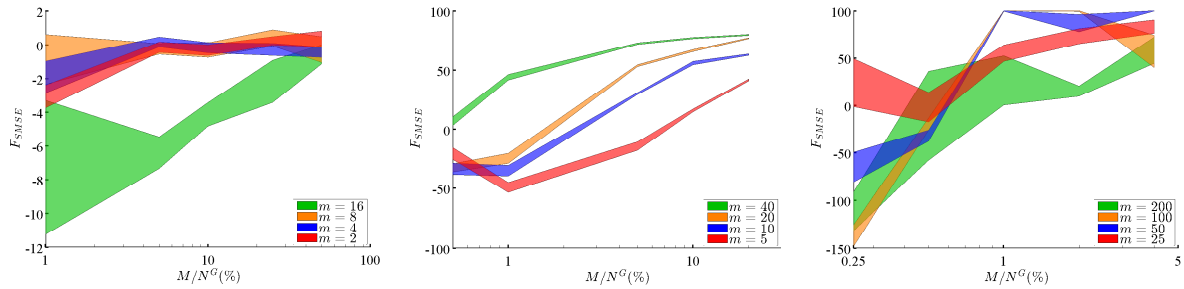


(a) F_{SMSE}, Abalone (b) F_{SMSE}, KIN40K (c) F_{SMSE}, SARCOS

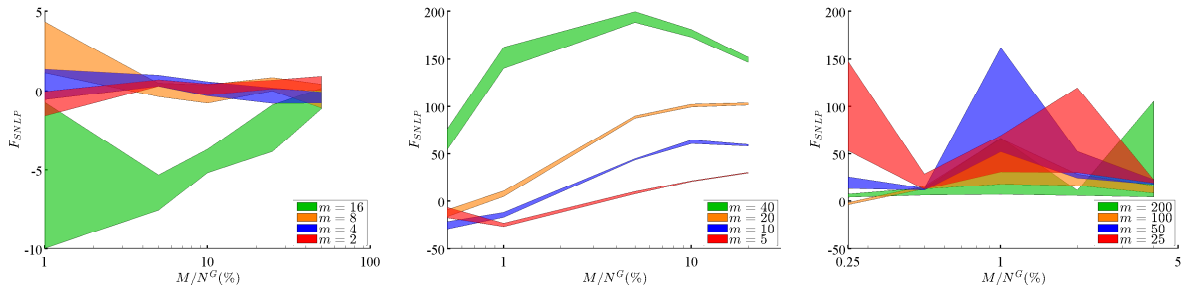


(d) F_{SNLP}, Abalone (e) F_{SNLP}, KIN40K (f) F_{SNLP}, SARCOS

Figure C.4. Error measures with separate vs. joint optimization of X_u and $\boldsymbol{\theta}$ for clustered data. Colored regions represent the mean value ± one deviation in uncertainty.



(a) F_{SMSE}, Abalone (b) F_{SMSE}, KIN40K (c) F_{SMSE}, SARCOS



(d) F_{SNLP}, Abalone (e) F_{SNLP}, KIN40K (f) F_{SNLP}, SARCOS

Figure C.5. Error measures with separate vs. joint optimization of X_u and $\boldsymbol{\theta}$ for randomly distributed data. Colored regions represent the mean value ± one deviation in uncertainty.

C.3 Random Subsets vs. Clustered Data

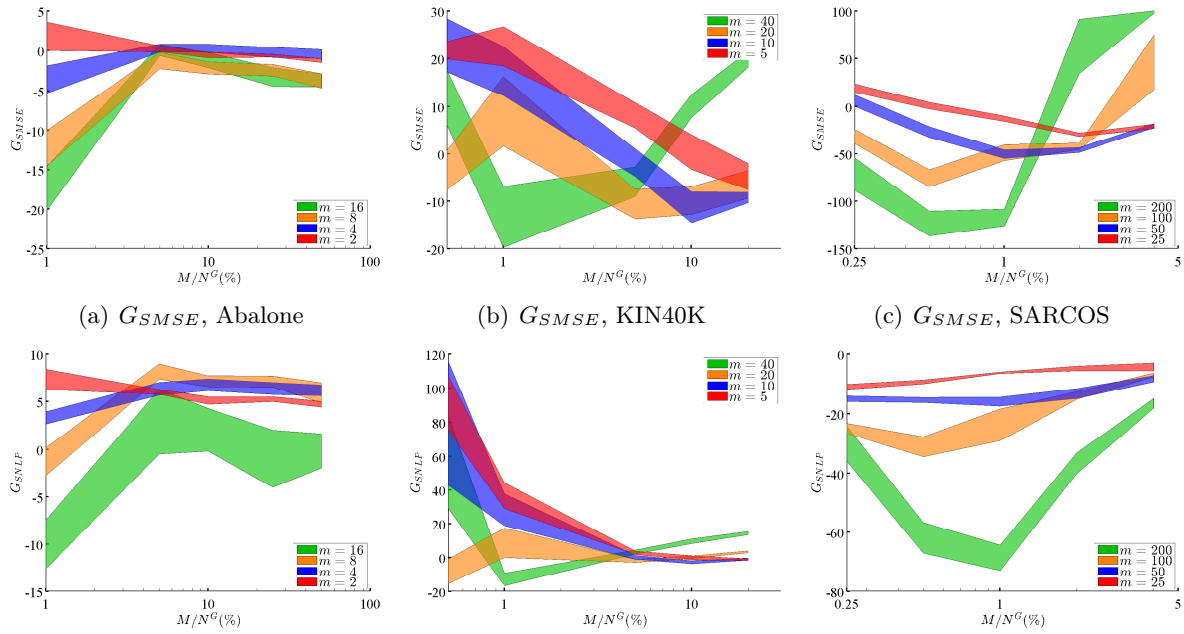
In previous sections, we tested the optimization of X_u against both clustered and unclustered data. We now evaluate the effects of clustering on predictive performance. Note that randomly distributing data amongst nodes effectively reduces our model to the BCM. Thus, we are testing whether the extra flexibility of our model in learning heterogeneous hyperparameters has ramifications on performance. According to the results of Sections C.1 and C.2, we separately optimize X_u and $\boldsymbol{\theta}$ for clustered data, and we do not optimize X_u for cases of randomly distributed data. We evaluate the percent change in performance due to clustering with the random distribution of data as a baseline:

$$G_{SMSE} = 100\frac{SMSE_R - SMSE_C}{SMSE_R}, \tag{C.3a}$$

$$G_{SNLP} = 100\frac{SNLP_R - SNLP_C}{|SNLP_R|}, \tag{C.3b}$$

where $SMSE_C$ and $SNLP_C$ are errors with clustered data, and $SMSE_R$ and $SNLP_R$ are errors with randomly distributed data. As with Equation C.2, we compare the means of all 5 trials: $SMSE_C$ is given by $x \pm \delta x$, where x is the mean $SMSE$ of all 5 trials with clustered data, and δx is the standard error. Uncertainties in G_{SMSE} and G_{SNLP} are computed by propagating errors in quadrature. Figure C.6 shows the results for all three datasets.

As before, the linearity of Abalone renders all changes negligible between cases of clustered and randomly distributed data. For KIN40K and SARCOS, we see that clustering generally has positive effects that decrease with M, except for very large m. As noted previously, large m with clustered data can lead to overfitting by local nodes, an effect that is most visible for small M. Larger M mitigates this problem and results in highly improved performance for the $SMSE$ with large m. Thus, overall we see that clustering generally improves performance away from the extreme limit of large m and small M. However, we recognize that none of these datasets is known for its heteroscedasticity. To truly test the effects of clustering and learning heterogeneous sets of hyperparameters, we require a heteroscedastic dataset which is known to contain widely disparate length scales and noise magnitudes at different regions of the input domain.



(a) G_{SMSE}, Abalone (b) G_{SMSE}, KIN40K (c) G_{SMSE}, SARCOS

(d) G_{SNLP}, Abalone (e) G_{SNLP}, KIN40K (f) G_{SNLP}, SARCOS

Figure C.6. Error measures with clustered vs. randomly distributed data. For the clustered case, we separately optimize X_u and $\boldsymbol{\theta}$; for the randomly distributed case, we choose X_u as random subsets of data. Colored regions represent the mean value $\pm$ one deviation in uncertainty.

Bibliography

[1] Bavelas, A. Communication patterns in task-oriented groups. *Journal of the acoustical society of America* (1950).

[2] Boyd, S., Ghosh, A., Prabhakar, B., and Shah, D. Gossip algorithms: Design, analysis and applications. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE* (2005), vol. 3, IEEE, pp. 1653–1664.

[3] Das, K. C., and Bapat, R. A sharp upper bound on the largest laplacian eigenvalue of weighted graphs. *Linear algebra and its applications 409* (2005), 153–165.

[4] Grant, M., and Boyd, S. Graph implementations for nonsmooth convex programs. In *Recent Advances in Learning and Control*, V. Blondel, S. Boyd, and H. Kimura, Eds., Lecture Notes in Control and Information Sciences. Springer-Verlag Limited, 2008, pp. 95–110. `http://stanford.edu/~boyd/graph_dcp.html`.

[5] Grant, M., and Boyd, S. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, Mar. 2014.

[6] Gupta, P., and Kumar, P. R. The capacity of wireless networks. *IEEE Transactions on Information Theory 46*, 2 (2000), 388–404.

[7] Horn, R. A., and Johnson, C. R. *Matrix analysis*. Cambridge university press, 2012.

[8] Meeds, E., and Osindero, S. An alternative infinite mixture of gaussian process experts. *Advances in Neural Information Processing Systems 18* (2006), 883.

[9] Nguyen, T., and Bonilla, E. Fast allocation of gaussian process experts. In *Proceedings of The 31st International Conference on Machine Learning* (2014), pp. 145–153.

[10] Nguyen-Tuong, D., Peters, J. R., and Seeger, M. Local gaussian process regression for real time online model learning. In *Advances in Neural Information Processing Systems* (2009), pp. 1193–1200.

[11] Olfati-Saber, R., Fax, J. A., and Murray, R. M. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE 95*, 1 (2007), 215–233.

[12] Peterson, L. L., and Davie, B. S. *Computer networks: a systems approach*. Elsevier, 2007.

[13] Quiñonero-Candela, J., and Rasmussen, C. E. A unifying view of sparse approximate gaussian process regression. *The Journal of Machine Learning Research 6* (2005), 1939–1959.

[14] Rasmussen, C. E., and Ghahramani, Z. Infinite mixtures of gaussian process experts. *Advances in neural information processing systems 2* (2002), 881–888.

[15] Rasmussen, C. E., and Williams, C. K. *Gaussian processes for machine learning*. MIT Press, 2006.

[16] Sabidussi, G. The centrality index of a graph. *Psychometrika 31*, 4 (1966), 581–603.

[17] Silverman, B. W. Some aspects of the spline smoothing approach to non-parametric regression curve fitting. *Journal of the Royal Statistical Society. Series B (Methodological)* (1985), 1–52.

[18] Sinha, A. Distributed consensus protocols in adaptive multi-agent systems. Undergraduate thesis, Princeton University, 2013.

[19] Snelson, E., and Ghahramani, Z. Variable noise and dimensionality reduction for sparse gaussian processes. *Proceedings of the 22nd Annual Conference on Uncertainty in AI.* (2006).

[20] Snelson, E., and Ghahramani, Z. Local and global sparse gaussian process approximations. In *International Conference on Artificial Intelligence and Statistics* (2007), pp. 524–531.

[21] Tam, S. M. On covariance in finite population sampling. *Journal of the Royal Statistical Society. Series D (The Statistician) 34*, 4 (1985), pp. 429–433.

[22] Tanenbaum, A. S., and Wetherall, D. J. *Computer Networks*. Prentice Hall, 2010.

[23] Titsias, M. Variational learning of inducing variables in sparse gaussian processes. In *Twelfth International Conference on Artitifical Intelligence and Statistics* (April 2009).

[24] Trefethen, L. N., and Bau III, D. *Numerical linear algebra*, vol. 50. Siam, 1997.

[25] Tresp, V. A bayesian committee machine. *Neural Computation 12*, 11 (2000), 2719–2741.

[26] Young, G. F., Scardovi, L., and Leonard, N. E. Robustness of noisy consensus dynamics with directed communication. In *Proceedings of the 2010 American Control Conference* (2010), pp. 6312–6317.